HAAGA-HELIA
ammattikorkeakoulu

Ohjelmistotestauksen suunnittelu -
Case: A-lehdet Oy:n laskujen tulostusohjelma

Eija Rauhala

Tietojenkäsittelyn koulutusohjelma

2010

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

Tekijät Eija Rauhala	**Ryhmä tai aloitusvuosi** 2007
Opinnäytetyön nimi Ohjelmistotestauksen suunnittelu – Case: A-lehdet Oy:n laskujen tulostusohjelma	**Sivu- ja liitesivumäärä** 59 + 43
Ohjaaja tai ohjaajat Raine Kauppinen	

Opinnäytetyön tavoitteena on selvittää ohjelmistotestauksen osuutta ohjelmistoprojektissa. Ohjelmistotestaukseen tutustumisen kautta päätavoitteena on luoda testauksen suunnittelumallit, joita voimme A-lehdet Oy:ssä tulevissa ohjelmistoprojekteissa hyödyntää.

Teoriaan ja aikaisempiin tutkimuksiin tutustumalla on tutkittu ohjelmiston ja testauksen laatua, testauksen sijoittumista ohjelmistoprojektiin, testaustasoja, testausmenetelmiä ja testaukseen liittyviä dokumentteja.

Opinnäytetyössä on tutkittu menetelmiä siihen liittyen, miten testauksen avulla saavutetaan laadukas ohjelmisto tehokkaasti ja mahdollisimman pienin kustannuksin. Näihin menetelmiin kuuluvat esimerkiksi testitapausten valinta, testauksen riittävyyden arviointi, testauksen tehokkuuden arviointi ja testauksen dokumentointi. Kaikkia, kuten staattisen mallin mukaisia testausmenetelmiä ei aina mielletä testaukseen kuuluviksi, mutta testauksen alussa käytettynä ne pienentävät projektin kustannuksia ja ovat hyvä virheiden etsimiskeino.

Testitapausten suunnittelu tulee aloittaa ohjelmistotuotantoprosessissa jo määrittelyvaiheessa. Aikaisessa vaiheessa löydetyt virheet vähentävät virheenkorjauksista aiheutuvia kustannuksia. Testauksen suunnittelun ja testauksen tulee edetä ohjelmistoprojektin kanssa rinnakkain.

Testaussuunnittelu eri testaustasoilla, joita ovat moduuli-, integrointi-, järjestelmä- ja hyväksymistestaus, on tämän opinnäytetyön päätavoite. Testaussuunnitelmia on toteutettu kaksi: moduulitestaussuunnitelma ja testaussuunnitelma, missä on yhdistettynä integraatio-, järjestelmä- ja hyväksymistestaus. Testaussuunnitelmat on tehty laskujen tulostuksen muutosprojektia varten. Varsinainen testaus ja sen tuloksiin liittyvät dokumentit eivät kuulu tähän opinnäytetyöhön.

Asiasanat
Testaussuunnitelma, ohjelmistotestaus, testaustasot, testausmenetelmät, testitapaukset

Abstract

HAAGA-HELIA
ammattikorkeakoulu

Degree programme

Authors Eija Rauhala	**Group or year of entry** 2007
The title of thesis Software Test Design - Case: A-lehdet Oy's invoice printing program	**Number of pages and appendices** 59 + 43
Supervisors Raine Kauppinen	

The objective of this thesis is to examine a software testing role in a software project. The main objective is to create a test design models, which we can utilized in future software projects in A-lehdet Oy.

The methods used in this study were reading theory and previous studies. The focus has been in software quality and testing quality, where the testing takes place in a software project, test levels, test methods and the documents of testing.

In this thesis has been studied methods, how the testing will achieve a high quality software efficiently and at minimum cost. These matters include, for example the selection of test cases, testing the adequacy of the evaluation, testing the effectiveness of the evaluation and testing documents. The study showed that static test methods are not always regarded as part of the testing, but using at the beginning of testing, they reduce the cost of the project and are a good way to find errors.

According to the study test case design should be started already in a specification phase in software production process. Errors which are found at early stage reduce the error correction costs. Test design and testing should proceed at the same time with the software project.

Test plan in different test levels, which are module, integration, system and acceptance, is the main objective of this thesis. Test plans have been made two: the module test plan and test plan, which is a combination of integration, system and acceptance testing. Test plans have been made for invoice printing project. Actual testing and the documents of the tests results do not include in this thesis.

Key words
Test plan, software testing, testing levels, test methods, test cases

Sisällys

1 Johdanto

Opinnäytetyö antaa yleiskuvan siitä, mitä ohjelmistotestauksessa tapahtuu. Ohjelmistotestauksen eri vaiheet suunnittelusta testitapausten tekemiseen ja testauksen raportointiin käydään läpi. Opinnäytetyön tarkoituksena on tutustua ohjelmistotestaukseen ja luoda testauksessa käytettävä suunnittelumalli, jota yrityksessämme voidaan hyödyntää. Ohjelmistotestaus pitää sisällään useita eri asioita ja opinnäytetyöni tavoitteena on selvittää

- ohjelmistotestauksen käsitettä,
- ohjelmiston ja testauksen laatua,
- ohjelmistotestauksen sijoittumista ohjelmistoprojektiin,
- ohjelmistotestausprosessin vaiheet,
- ohjelmistotestauksen testausmenetelmiä,
- testaussuunnitelman tarkoitus,
- ohjelmistotestauksen onnistumiseen vaikuttavat tekijät sekä
- ohjelmistotestauksen suunnittelu- ja raportointimalleja.

Päätavoitteeni on luoda testauksen suunnittelumalli, jota voimme hyödyntää A-lehdet Oy:ssä tulevissa ohjelmistoprojekteissa. Tätä mallia on tarkoitus lähteä luomaan esimerkin kautta, joka liittyy laskutuksen tulostuksen uusimiseen. Tarkoituksenani on tehdä laskutuksen tulostuksessa tarvittavat testaussuunnitelmat: yleistestaussuunnitelma ja moduulitestaussuunnitelma. Henkilökohtaisella tasolla tavoitteenani on tutustua testauksen teoriaan, koota siitä hyvä yleiskäsitys ja hyödyntää näitä tietoja työssäni tulevaisuudessa.

Opinnäytetyössä tutustutaan ensimmäiseksi ohjelmistotestauksen käsitteeseen. Sekä ohjelmistolta että testaukselta vaaditaan hyvää laatua. Testauksen tavoitteena on tuottaa laadukas ohjelmisto, jossa ei ole virheitä. Laatuun tutustumisen yhteydessä paneudutaan syvällisemmin staattisista testausmenetelmistä tarkastuksiin. Tarkastuksilla voidaan vaikuttaa jo varhaisessa vaiheessa testauksen laatuun.

Perinteisessä ohjelmistoprojektissa testaus sijoittuu prosessin loppupäähän. Opinnäytetyössä lähdetään tutkimaan mihin testauksen tulisi ohjelmistoprojektissa sijoittua, jotta testauksesta ja itse ohjelmistoprojektista tulisi mahdollisimman kustannustehokas.

Testaussuunnitelma on hyvä lähtökohta lähteä suorittamaan testausta. Suunnitelman avulla saadaan selville ohjelmistoprojektin kriittiset kohdat. Opinnäytetyössä tutustutaan testaussuunnitelman tarkoitukseen, sen sisältöön ja tuotoksiin. Testaustasoihin tutustutaan yleisen V-mallin kautta. Testausmenetelmistä käydään läpi yleisimmät testausmenetelmät ja testitapausten valintakriteerit. Lisäksi tutustutaan testauksen tehokkuuden arviointimenetelmiin.

Opinnäytetyössä tutustutaan perusteellisesti testauksen suunnittelu- ja raportointimalleihin tarkoituksena hyödyntää näitä malleja tulevissa ohjelmistoprojekteissa yrityksessämme. Testauksen suunnitteluun ja raportointiin liittyviä dokumentteja on määritelty eri tarkkuustasoille alkaen yleisestä testaussuunnitelmasta päätyen yksittäisten testitapausten suunnitteluun ja testitapausten tulosten raportointiin. Käyn nämä dokumentit läpi. Tarkoituksena on, että näistä mallipohjista voisi jatkossa luoda eri ohjelmistoprojekteihin sopivat testaussuunnitelmat ja –raportit. Lopuksi esittelen testaussuunnitelman ja moduulitestaussuunnitelman, joka liittyy laskujen tulostusohjelmaan ja on tehty näiden opintojen pohjalta.

Menetelminä käytän teoriaan tutustumista kirjallisuuden, aikaisempien tutkimusten ja internetistä löytyvän materiaalin pohjalta. Teorian pohjalta kirjoitan testaussuunnitelman ja moduulitestaussuunnitelman mukaillen IEEE standardin 829-1998 mallipohjaa.

1.1 Rajaukset

Itse testauksen toteuttaminen ei kuulu tämän opinnäytetyön kokeelliseen osaan. Testaukseen liittyvät testiraportit, testauksen tapahtumaraportti ja testauksen yhteenvetoraportti, jäävät näin ollen myös kokeellisesta osasta pois.

1.2 Keskeiset käsitteet

Dynaaminen analyysi	Ohjelmiston testaaminen ohjelmakoodia suorittamalla
Harmaalaatikkotestaus	Musta- ja valkolaatikkotestauksen välimuoto, jossa käytetään hyväksi tietoa ohjelman toteutusperiaatteista
Hyväksymistestaus	Loppukäyttäjän suorittama testaus, jossa arvioidaan tuotteen kykyä vastata asiakasvaatimuksiin
IEEE standardi 829-1983	Standardi ohjelmistojen testausdokumenteille

Integrointitestaus	Testaus, jonka tarkoituksena on testata moduulien välisten rajapintojen toimintaa
Järjestelmätestaus	Testaus, jonka tarkoituksena on testata koko järjestelmän toimintaa
Katselmus	Menetelmä, jolla todetaan jonkin ohjelmistotuotantoprosessin vaiheen päättyminen
Kelpoistaminen	Menetelmä, jolla tutkitaan tuotteen sopivuus käyttötarkoitukseensa
Lasilaatikkotestaus	Testausmenetelmä, jossa nähdään testattavan kokonaisuuden sisäinen toiminta ja rakenne eli koodi. Tunnetaan myös nimellä valkolaatikkotestaus
Läpikäynti	Menetelmä, jolla varmistetaan, että tuote vastaa vaatimuksia ja määritelmiä kooditasolla. Englanninkielinen nimi walkthrough
Moduulitestaus	Testausvaihe, jossa kohteena on vain yksittäinen moduuli
Mustalaatikkotestaus	Testausmenetelmä, joka perustuu testattavan kokonaisuuden ulkoiseen toimintaan (syötteisiin ja tulosteisiin)
Regressiotestaus	Testausmenetelmä, jolla todennetaan ohjelmistoon tehdyn korjauksen toimivuus ja varmennetaan, että uusia vikoja ei ole syntynyt muuttumattomiin ohjelmiston osiin korjauksen myötä
Staattinen analyysi	Ohjelmiston testaaminen ilman ohjelmakoodin suorittamista
STEP metodologia	The Systematic Test and Evaluation Process Metodologia, joka näkee testauksen prosessina alkaen heti määrittelyvaiheen yhteydessä
Testaussuunnitelma	Suunnitelma, jonka pohjalta ohjelmistotestaus voidaan viedä organisoidusti läpi
Tarkastus	Menetelmä, jolla varmistetaan, että tuote vastaa vaatimuksia ja määritelmiä
Vesiputousmalli	Perinteinen ohjelmistotuotantoprosessia kuvaava elinkaarimalli

2 Ohjelmistotestaus

Ohjelmistotestausta suoritetaan, jotta varmistuttaisiin ohjelmistojen toimivuudesta ja että ohjelmisto täyttää sille asetetut vaatimukset eikä siinä ole virheitä. Testauksessa varmistutaan, että ohjelmisto täyttää asiakkaiden ja käyttäjien tarpeet, että ne ovat nopeita ja käyttäjäystävällisiä. Ohjelmistolta ja testaukselta vaaditaan siis hyvää laatua. Graigin ja Jaskielin (2002, 6) mukaan monissa yrityksissä testausta suoritetaan vasta ohjelmistoprojektin loppuvaiheessa, kun koodi on jo kirjoitettu. Kustannukset puolestaan kasvavat mitä myöhemmin virheet löydetään ja korjataan. Ohjelmistotestauksen tulisi kuulua olennaisena osana ohjelmistotuotantoprosessia vaatimusmäärittelystä alkaen, jotta kustannukset eivät tulisi kohtuuttomiksi.

Testauksen työvaiheisiin kuuluvat testauksen suunnittelu, testiympäristön luominen, testin suorittaminen, testitulosten tarkastelu ja raportointi. Haikalan ja Märijärven (2006, 283) mukaan näihin työvaiheisiin ja niihin liittyviin virheiden etsimiseen ja korjaukseen kuluu tyypillisesti yli puolet ohjelmistoprojektin resursseista.

Testauksen kohteena ovat yleensä ohjelmiston erilaiset toiminnot kuten tietojen lisäys, poisto tai päivitys. Testauksen kohteena voi olla myös suorituskyky, jolloin testauksessa kiinnitetään huomiota suuriin tapahtumamääriin. Myös turvallisuutta voidaan testata jolloin tarkastetaan, että oikeat käyttäjät pääsevät käsiksi oikeisiin tietoihin. Konfiguraatiotestauksia suoritetaan ohjelmistoille, jotka on suunniteltu toimimaan erilaisissa laite- ja käyttöjärjestelmäympäristöissä. (Rajamäki 2000, 4.)

Yksinkertaistettuna esimerkkinä testauksesta voimme ajatella ohjelmalle annettavaa syötettä X, josta ohjelma tuottaa tulosteen Y. Testaustulos riippuu syötteen lisäksi järjestelmän sisäisestä tilasta. Sisäinen tila tarkoittaa mm. ohjelman muuttujien ja rekisterien arvoja sekä levylle talletettuja tietoja. Testin onnistuminen edellyttää, että tulos Y on oikea ja että sisäinen tila on muuttunut oikein. Testausta varten on oltava käsitys ohjelman syötteistä ja oikeasta lopputuloksesta. Tätä varten tarvitaan spesifikaatio, jonka perusteella testitapaukset voidaan suunnitella ja voidaan päätellä oikeat lopputulokset. (Haikala & Märijärvi 2006, 285.)

2.1 Ohjelmistotestauksen määritelmä

Ohjelmistotestauksen perinteinen määritelmä on virheiden suunnitelmallinen etsiminen. Testaukseen valitaan usein ilman sen kummempia suunnitelmia syöttöaineisto ja tavoitteena on ennemminkin osoittaa ohjelman toimivuus kuin virheiden löytyminen. Nykyään testaus määritellään ohjelmiston laadun mittaamisen ja parantamisen näkökulmasta. (Haikala & Märijärvi 2006, 284.)

Graigin ja Jaskielin (2002, 4 - 6) mukaan ”Testing is a concurrent lifecycle process of engineering, using and maintaining testware in order to measure and improve the quality of the software being tested”. Eli korostetaan sekä testauksen tuottamien mittareiden tärkeyttä että ohjelmiston laadun parantamista. Tämä tunnetaan ehkäisevän testauksen (preventive testing) käsitteenä, jolloin ohjelmiston vaatimusmäärittelyn pohjalta kirjoitetaan testitapauksia. Koodaajan on huomattavasti helpompi aloittaa työskentely, kun hänellä on vaatimusmäärittelyn lisäksi valmiita testitapauksia. Ehkäisevässä testauksessa käytetään menetelmänä esimerkiksi tarkastuksia (inspections), joista kerrotaan enemmän luvussa 2.4 Laadunvarmistus. Koska testauksella pyritään hyvään laatuun, seuraavaksi hieman, mitä laadulla tarkoitetaan.

2.2 Ohjelmiston laatu

Ohjelmiston laadulle asetetut vaatimukset voivat vaihdella suurestikin käyttötarkoituksen mukaan. Esimerkiksi suunniteltaessa sairaalaan teho-osaston valvontalaitteistoa, ovat laatuvaatimukset huomattavasti korkeammat kuin suunniteltaessa ohjelmistoja, jotka eivät vaaranna ihmishenkiä. (Enqvist ym. 2001, 3) mukaan hyvälaatuisen ohjelmiston tulisi

- vastata tarkoitustaan ja määritystään,
- olla helppokäyttöinen ja riittävän yksinkertainen,
- olla helposti ylläpidettävä ja päivitettävä,
- olla turvallinen sekä
- olla mahdollisimman suorituskykyinen.

Ohjelmistotestauksella pyritään varmistamaan ohjelmiston laatu. Tavoitteena on tuottaa kustannustehokkaasti toivottu lopputulos, valmis ohjelmisto. Kustannustehokkuuteen ohjelmistojen laadussa päästään silloin, kun testauksen suunnittelu aloitetaan mahdollisimman varhaisessa vaiheessa yhtä aikaa itse ohjelmistoprojektin kanssa.

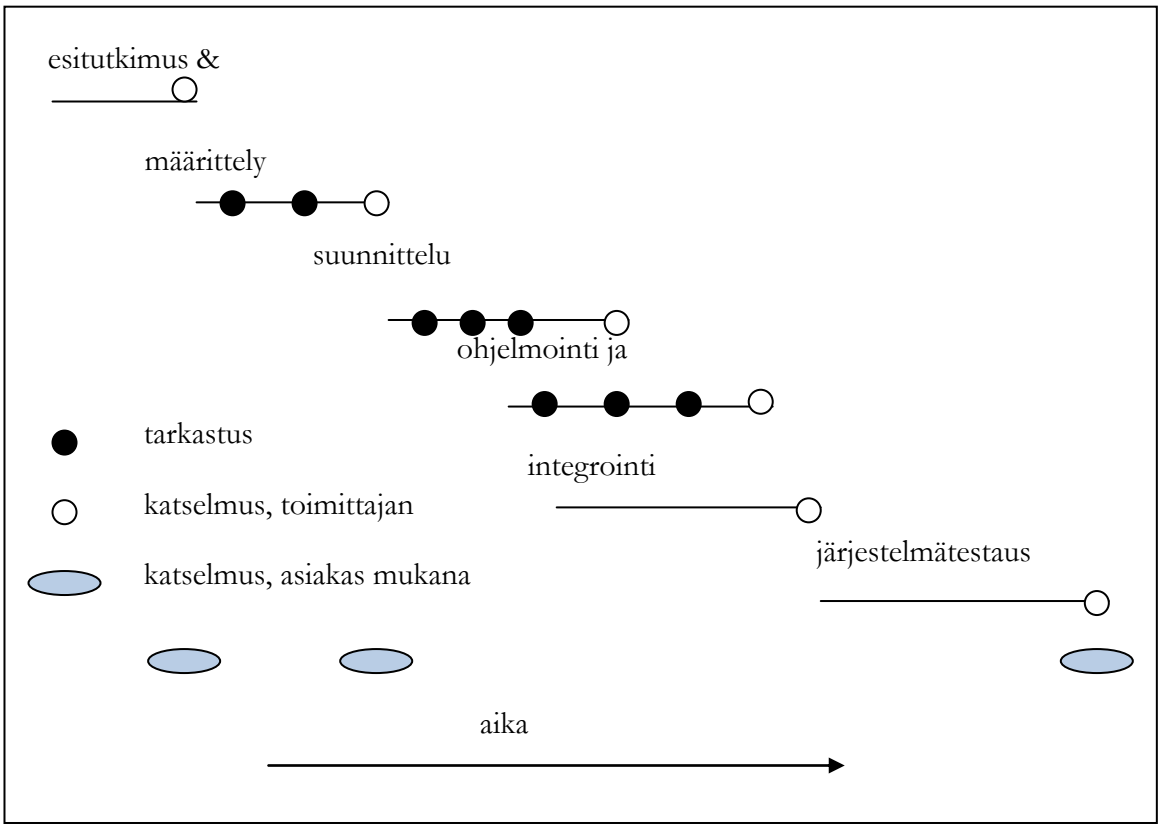
2.3 Testauksen laatu

Testauksen laatua voidaan mitata sillä, miten kattavasti testaus on suoritettu. Täydellisestä kattavuudesta puhutaan silloin, kun ohjelmiston kaikki mahdolliset reitit on tutkittu kaikilla mahdollisilla syötteillä. Tämä on käytännössä mahdotonta toteuttaa. Testausprosessi on sidottu käytettävissä oleviin resursseihin kuten raha, aika ja henkilöstöresurssit. Kattavuudesta on kerrottu luvussa 4.2 lasilaatikkotestauksen yhteydessä ja luvussa 5.3 Testauksen tehokkuuden arvioinnin yhteydessä. (Enqvist ym. 2001, 3.)

2.4 Laadunvarmistus

Laadunvarmistuksen yhteydessä puhutaan **katselmuksesta** (tekninen katselmus, technical review) sekä **todentamisesta ja kelpoistamisesta** (verification and validation). Katselmuksessa todetaan jonkin vaiheen päättyminen. Siinä varmistetaan, että asetetut kriteerit on täytetty ja vaaditut asiat suoritettu. Katselmuksia pidetään esimerkiksi määrittely- tai suunnitteluvaiheen päättyessä. Laadunvarmistamisen lisäksi toimenpiteillä saavutetaan projektin etenemisen näkyvyys. Myös tietämys kohteena olevasta ohjelmistosta lisääntyy, on sitten kyseessä uusi ohjelmisto tai ohjelmistoon tehtävät muutokset.

Todentaminen ja kelpoistaminen puolestaan ovat ohjelmiston laadun tarkastelua. Todentamisessa (objektiivinen laatu) esimerkiksi järjestelmätestauksen yhteydessä verrataan tuotetta sen toiminnalliseen määrittelyyn. Todentamismenettelyjä ovat **tarkastusmenettely** (inspection) ja **läpikäynti** (walkthrough). Tarkastuksissa dokumentteja verrataan vaatimusmäärittelyihin ja läpikäynnissä tarkastus tehdään yleensä koodille. Kelpoistamisessa tarkastellaan tuotteen sopivuutta sen käyttötarkoitukseen (subjektiivinen laatu). Katselmusten ja tarkastusten erona voidaan pitää sitä, että katselmuksessa havaittu virhe on merkki huonosti tehdystä työstä, kun taas tarkastuksessa havaittu virhe on merkki hyvin tehdystä työstä. Virheiden korjaukset maksavat huomattavasti enemmän, jos ne havaitaan katselmuksessa. Virheiden kustannusvaikutuksista projektiin on enemmän tarkastusten yhteydessä tässä luvussa. Kuviosta 1 selviää tarkemmin miten katselmukset ja tarkastukset projektiin ajoittuvat. (Haikala & Märijärvi 2006, 267–269.)



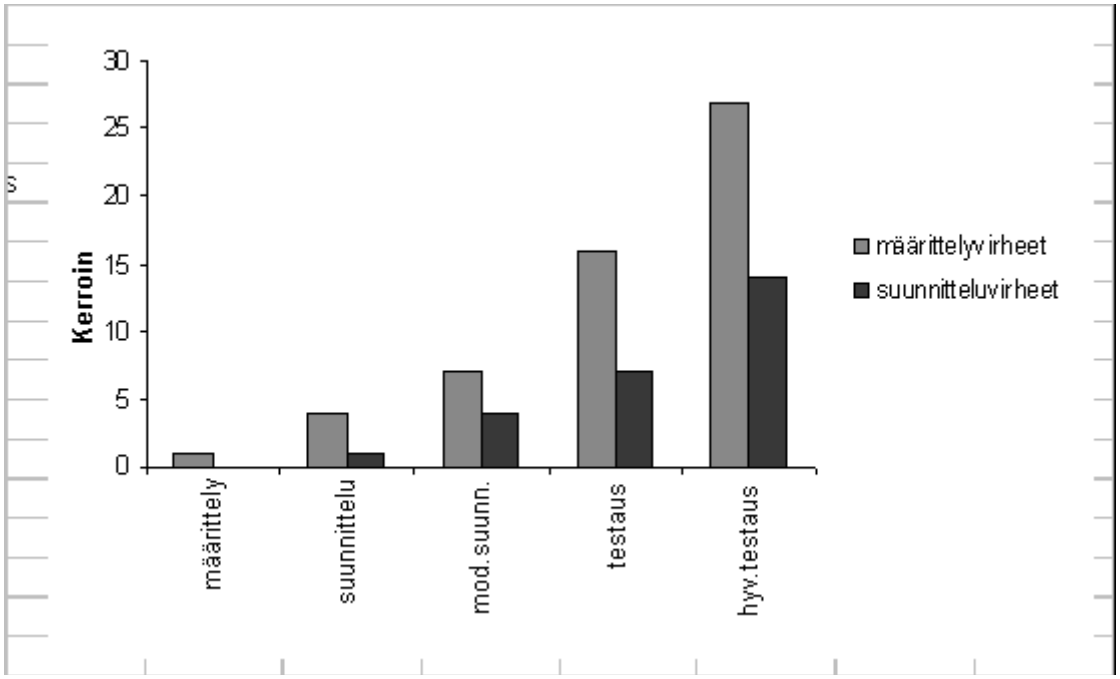
Kuvio 1. Projektin katselmukset ja tarkastukset (Haikala & Märijärvi 2006, 52)

Tarkastukset

Tässä käytetään termiä tarkastus, mutta samaa menetelmää voi käyttää myös katselmukseen ja läpikäyntiin. Tarkastusten oppi-isä on Fagan, joka kehitti menetelmän IBM:llä jo 1970-luvulla. Tarkastusten avulla pyritään auttamaan ohjelmistoprojektin elinkaaren (määrittely-suunnittelu-toteutus-testaus-käyttöönotto) etenemistä niin, että virheet löydettäisiin mahdollisimman varhaisessa vaiheessa ja eteneminen olisi näkyvää. Tarkastukset kannattaisi tehdä ainakin määrittelydokumentille, projektisuunnitelmalle, suunnitteludokumenteille ja testiraporteille. Tarkastuksen vaiheita ovat tarkastuksen suunnittelu, tarkastustilaisuus, virheiden korjaaminen ja jälkiseuranta. Koska tarkastusten suunnittelua olisi hyvä tehdä jo projektisuunnittelun yhteydessä, tämä täytyisi huomioida projektisuunnitelmaa tehtäessä varaamalla aikaa tarkastuksiin. (Haikala & Märijärvi 2006, 269–270.)

Tarkastettava materiaali jaetaan osallistujille ajoissa, jotta he ehtivät tutustua tarkastettavaan aiheeseen ja löytää mahdolliset virheet tai ongelmakohdat. Itse tarkastustilaisuudessa dokumentti käydään läpi ja kaikki löydetyt virheet kirjataan ylös. Sovitaan myös siitä, kuka korjaa löydetyt virheet ja miten jälkiseuranta toteutetaan. (Enqvist ym. 2001, 8.)

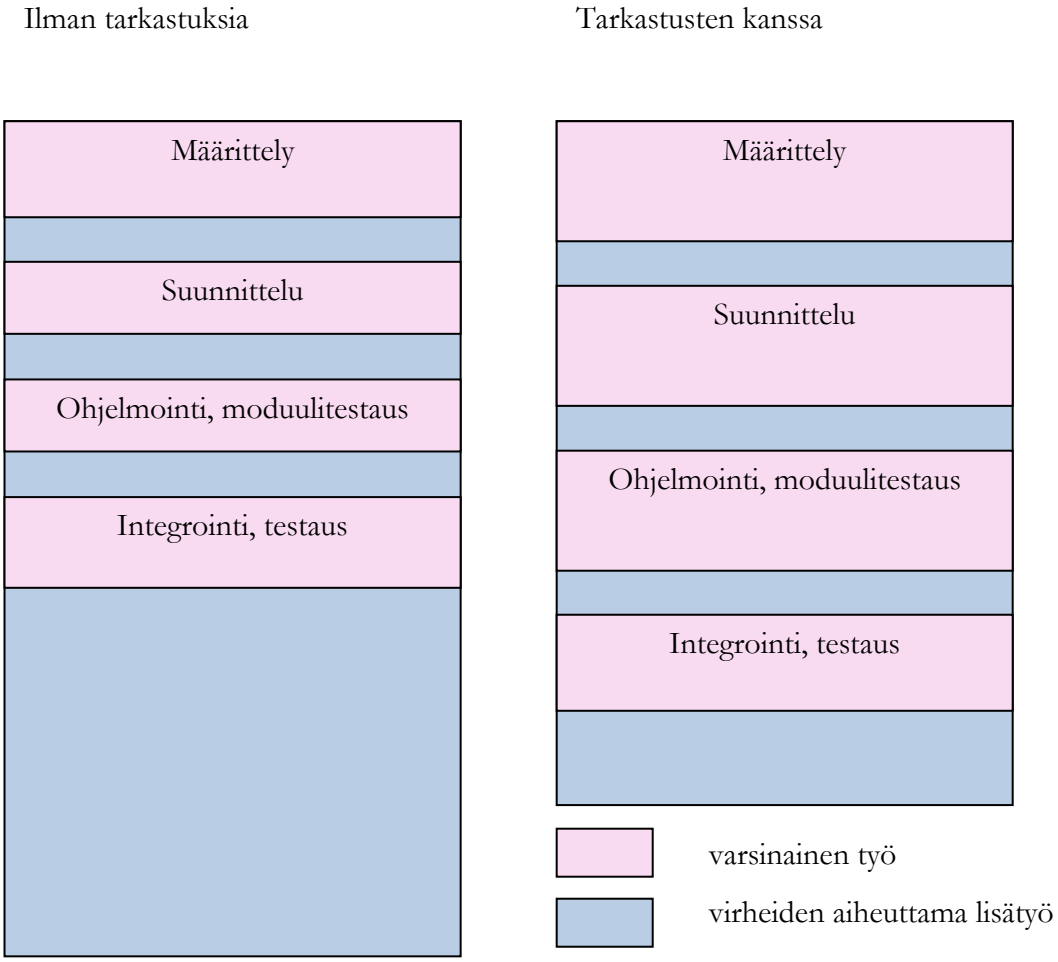
Tarkastukset ovat mm. testausta tehokkaampi virheiden etsintäkeino. Haikalan ja Märijärven (2006, 275) mukaan määrittelyvirheen korjaaminen suunnitteluvaiheessa on neljä kertaa kalliimpaa kuin heti määrittelyn yhteydessä. Moduulisuunnitteluvaiheessa kerroin on 7 jne. Suunnitteluvirheet vaikuttavat samalla tavalla. Kuvio 2 havainnollistaa virhekustannusten kertoimia suhteessa siihen, milloin virheet havaitaan. Tarkastuksilla voidaan löytää 80 % kaikista virheistä. Tarkastuksiin kuluu yleensä 5-15 % projektin kokonaistyöajasta. Muita hyviä puolia tarkastuksissa on, että tarkastuksiin tuotetaan mahdollisimman valmista materiaalia. Kukaan ei halua päästää käsistään keskeneräistä työtä muiden tarkastettavaksi. Toinen erittäin tärkeä tekijä on tiedonkulku, tarkastukset toimivat erinomaisena tiedotus- ja koulutustilaisuutena.



Kuvio 2. Virhekustannusten kertoimia (Haikala & Märijärvi 2006, 275, teoksessa Jalote 1999)

Kuvio 3 puolestaan havainnollistaa virheiden aiheuttaman lisätyön osuutta eri vaiheissa ilman tarkastuksia tai tarkastusten kanssa. Vasemmanpuoleinen pylväs kertoo, miten paljon virheiden korjaamiseen kuluu resursseja, jos tarkastuksia ei käytetä. Tällöin virheiden korjaus tapahtuu

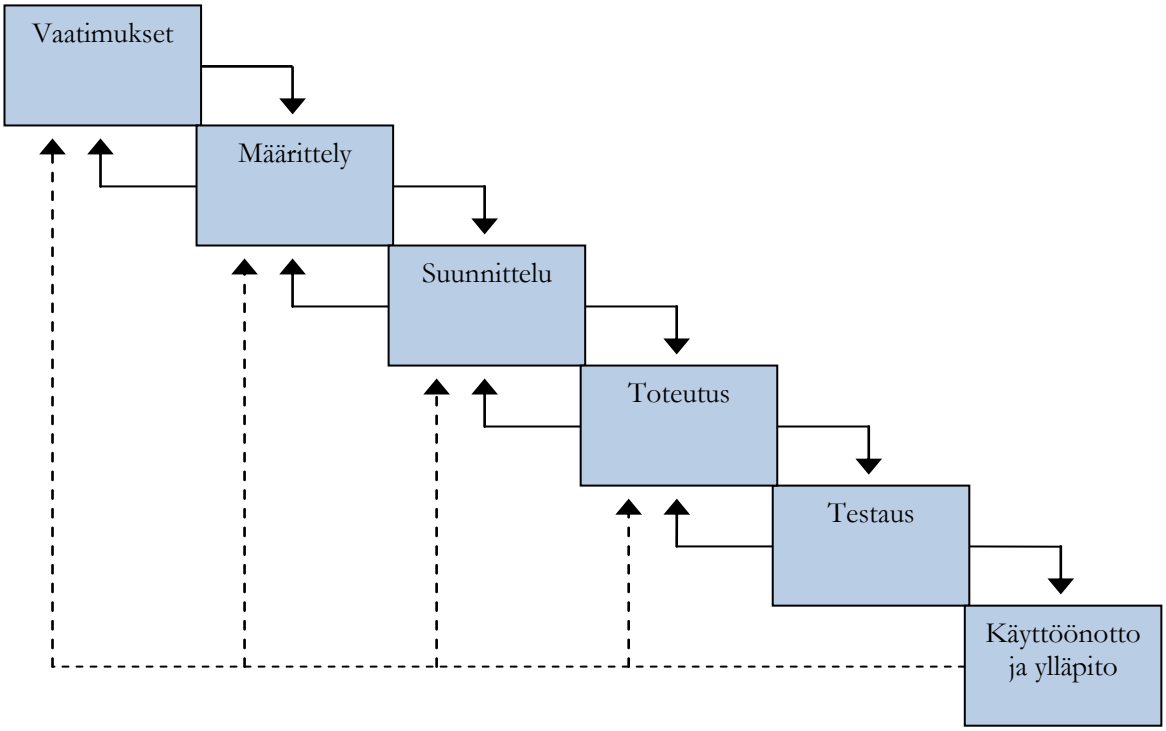
testausvaiheessa, joka aiheuttaa enemmän lisätyötä kuin jos olisi käytetty tarkastuksia. Oikeanpuoleisesta pylväästä näkyy, että kustannukset voivat alkuvaiheessa kasvaa tarkastusten seurauksena, mutta testausvaiheessa kustannukset ovat huomattavasti pienemmät. (Haikala & Märijärvi 2006, 276.)



Kuvio 3. Virheiden aiheuttaman lisätyön osuus ei vaiheissa (Haikala & Märijärvi 2006, 276)

3 Ohjelmistoprojektit ja testaus

Ohjelmistoprojektista erotellaan yleisesti seuraavat vaiheet: määrittely, suunnittelu, toteutus ja testaus sekä näitä vaiheita seuraavat käyttöönotto ja lopuksi ohjelmiston ylläpito. Yleisimmin kuvattu vaihejakomalli on vesiputousmalli, jonka perinteinen versio on kuvattuna kuviossa 4. Mallista löytyy eri versioita, mutta yleensä niistä löytyvät määrittely-, suunnittelu- ja toteutusvaiheet. Määrittelyvaihetta edeltää usein tarvekartoitus. Perinteisen vesiputousmallin hyvinä puolina voidaan pitää sitä, että kaikki voimavarat keskittyvät aina yhteen vaiheeseen kerrallaan ja seuraavaan vaiheeseen mentäessä edellisen vaiheen tuotokset ovat käytettävissä.



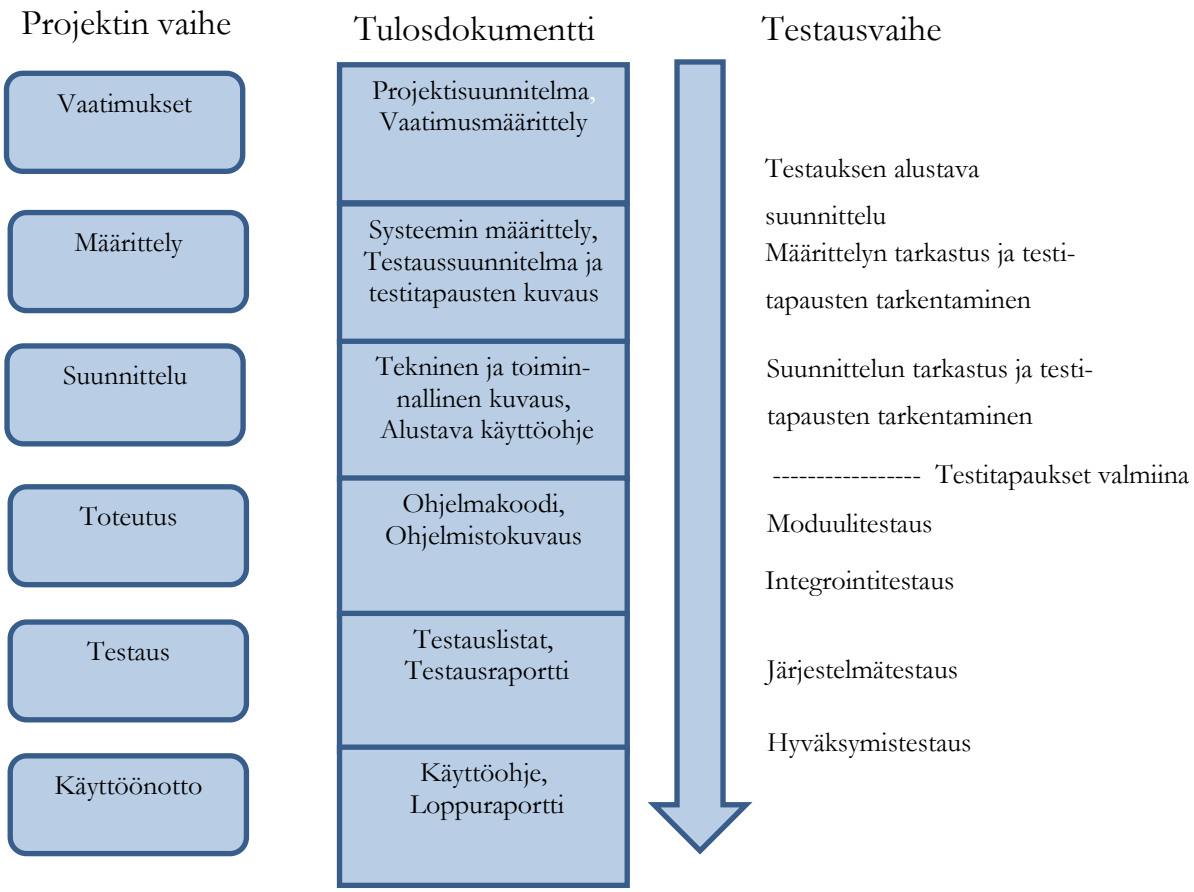
Kuvio 4. Perinteinen vesiputousmalli ohjelmistotuotantoprosessin vaiheista

Kuten kuviosta 4 käy ilmi, ongelmakohtana voidaan pitää varsinaisen testauksen sijoittumista vaihejakomallissa prosessin loppuun ennen käyttöönottoa. Vasta tässä vaiheessa aloitettuna testaus tuottaa huomattavasti enemmän kustannuksia projektiin, kuin että testauksen suunnittelu aloitetaan heti projektin alussa määrittelyvaiheen alkaessa. Toinen ongelma testaajien kannalta on, että yleensä testausvaiheessa ohjelmistoprojektilla on jo kiire ja testattava aineisto tulee usein myöhässä. Testausajan jäädessä liian lyhyeksi, ohjelmiston laatu kärsii. (Craig & Jaskiel 2002, 7–8.)

3.1 Testauksen sijoittuminen ohjelmistoprojektiin

Kuten aikaisemmin on jo tullut ilmi, tulisi testaus aloittaa mahdollisimman varhaisessa vaiheessa, mieluiten heti määrittelyvaiheessa. Graig ja Jaskiel (2002, 10–21) puhuvat kirjassaan STEP metodologiasta (The Systematic Test and Evaluation Process). Se perustuu IEEE standardiin 829-1983, joka on standardi ohjelmistojen testausdokumenteille (Standard for Software Test Documentation). Se on ensimmäisen kerran esitelty 1985. Metodologia perustuu ajatukselle, joka näkee testauksen prosessina, joka menee rinnakkain ohjelmistokehityksen tai ohjelmiston ylläpidon kanssa. Metodologia jakautuu samoihin työvaiheisiin, joista Haikala ja Märijärvikin (2006, 283) ovat teoksessaan kertoneet eli testauksen suunnittelu (yleinen testaussuunnitelma ja yksityiskohtainen testaussuunnitelma), testiympäristön luominen, testin suorittaminen ja testitulosten tarkastelu. STEP metodologian suurimmat erot perinteiseen testaukseen ovat testauksen aloittaminen ajoissa heti määrittelyvaiheen yhteydessä ja tätä kautta riskien parempi hallinta. Myös dokumentointi on parempaa ja tekee testauksen näkyväksi.

Riippuen projektin suuruudesta, onko kysymyksessä uusi ohjelmisto vai ohjelmiston ylläpito, eri vaiheisiin käytettävät panokset voivat vaihdella suurestikin. Oheisessa kuviossa 5, Jäntti (2003, 21) on kuvannut projektin eri vaiheet, sekä miten testaus sijoittuu suhteessa projektin etenemiseen. Oheisen mallin mukaan edeten voidaan puhua ennalta ehkäisevästä testauksesta. Testauksessa voidaan erotella neljä eri testaustasoa: yksikkö-, integrointi-, järjestelmä- ja hyväksymistestaus. Eri testaustasoista kerrotaan enemmän luvussa 3.2.

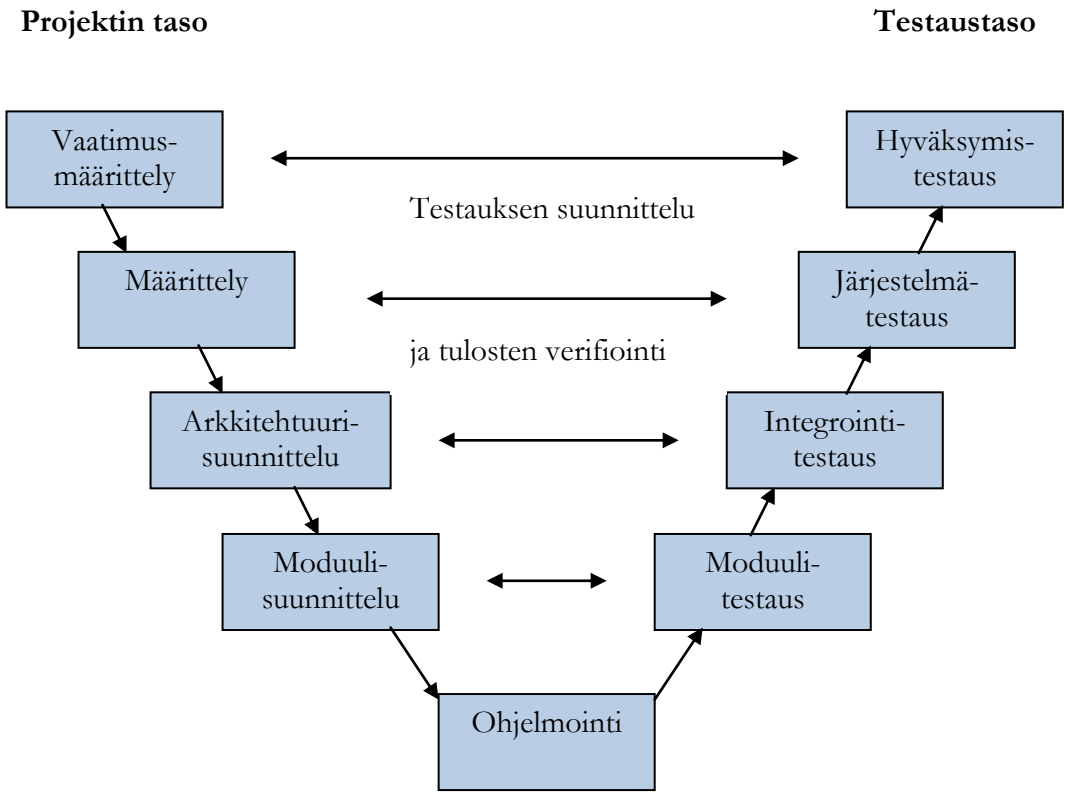


Kuvio 5. Testauksen suunnittelu ohjelmistotuotannon eri vaiheissa (Jäntti 2003, 21)

3.2 Testaustasot

Yleisin käytetty testausmalli on V-malli. Kuviossa 6 on kyseinen malli esiteltynä. Mallin mukaisesti testauksen suunnittelu tehdään testaustasoa vastaavalla suunnittelutasolla. Hyväksymistestaus suunnitellaan vaatimusmäärittelyvaiheessa, järjestelmätestaus suunnitellaan määrittelyvaiheessa, integrointitestaus arkkitehtuurisuunnitteluvaiheessa ja moduulitestaus moduulisuunnitteluvaiheessa. Testaukseen kuuluvat vaiheet ovat suunnittelu, suoritus ja dokumentointi. (Haikala & Märijärvi 2006, 288.)

Joissakin V-malleissa ei hyväksymistestaustasoa esitetä. Otan kuitenkin tähän malliin mukaan kaikki neljä tasoa, koska esityksen tarkoituksena on kattaa testaus kokonaisuudessaan.



Kuvio 6. Testauksen V-malli (Jäntti 2003, 13)

Itse testaus tapahtuu käänteisessä järjestyksessä eli vaikka hyväksymistestaus on ensimmäinen joka suunnitellaan, se testataan viimeisenä. Suunnittelu alkaa hyväksymistestauksesta, koska projektisuunnittelussa vaatimusmäärittely tehdään ensimmäisenä. Kun siirrytään testauksen suunnittelussa seuraavalle tasolle, suunnittelun perusteena ovat edellisen ja kyseisen tason suunnittelun määritykset, niin että moduulitestausta suunniteltaessa käytössä ovat moduulisuunnittelun, arkkitehtuurisuunnittelun, määrittelyn ja vaatimusmäärittelyn määritelmät. Kaikissa testaussuunnitelmissa määritellään kriteerit, milloin testaus voidaan aloittaa ja milloin siirrytään seuraavaan vaiheeseen. Testaustasot esitellään ylhäältä alaspäin, koska testien suunnittelukin aloitetaan sieltä eli hyväksymistestauksesta. (Craig & Jaskiel 2002, 101.)

Hyväksymistestaus

Hyväksymistestaus perustuu vaatimusmäärityksiin ja testauksen on tarkoitus näyttää toteen, että nuo vaatimukset on huomioitu. Testaussuunnitelman yksi päätarkoitus on määritellä millä kriteereillä ohjelmisto katsotaan hyväksytyksi. Testaussuunnitelman ja testitapausten on tarkoitus näyttää asiakkaalle tai hyväksyjälle, miltä ohjelmisto tai järjestelmä näyttää, kun se on val-

mis. Testausta voidaan siis pitää myös demo-tilaisuutena. Hyväksymistestaus on asiakkaan vastuulla. Yleensä hyväksymistestauksessa on läsnä it-henkilöstöä, asiakkaan edustajia tai organisaation sisäisissä projekteissa ohjelmiston käyttäjiä. Testaussuunnitelmaa tehtäessä on syytä kiinnittää huomiota, ettei sanasto ole liian teknistä. Hyväksymistestaussuunnitelmaa tehtäessä tarvitaan projektisuunnitelma, yleistestaussuunnitelma ja vaatimusmäärittelydokumentti. Testitapauksia suunniteltaessa vaatimukset otetaan huomioon niin, että yhtä vaatimusta testataan yhdessä tai useammassa testitapauksessa ja näin vaatimukset ovat jäljitettävissä. Hyväksymistestausta suoritettaessa tulisi testiympäristön vastata todellisuutta mahdollisimman hyvin. (Craig & Jaskiel 2002, 102–121.)

Järjestelmätestaus

Järjestelmätestauksessa koko järjestelmä (ohjelmisto, laitteisto, tietokanta) on tarkastelun kohteena. Järjestelmätestauksen suunnittelussa käytetään hyväksi määrittelydokumenttia, jossa on ohjelmiston toiminnallinen määrittely sekä asiakas- ja suunnitteludokumentteja, jos nämä kaikki ovat olemassa. Järjestelmätestauksen suunnittelu voidaan siis aloittaa heti kun määrittely-, asiakas- ja suunnitteludokumentaatio on valmis. Jos hyväksymistestaustasoa ei ole erikseen määritelty, se liitetään järjestelmätestaukseen. Järjestelmätestauksessa testataan myös järjestelmän ei-toiminnalliset ominaisuudet mm. kuormitustestit, luotettavuustestit ja asennustestit. Järjestelmätestaus pohjautuu integrointitestauksen tuloksiin. (Haikala & Märijärvi 2006, 288.)

Jos järjestelmätestaus aloitetaan ennen kuin integrointitestaus on lopetettu, voi järjestelmätestauksessa tulla ilmi virheitä, jotka olisi huomattu jo integrointitestausvaiheessa. Tällaisten virheiden korjaamiseen menee enemmän aikaa ja rahaa. Virheiden korjauksen jälkeen järjestelmätestausta joudutaan suorittamaan uudelleen eli tekemään ns. regressiotestausta. Järjestelmätestauksesta saadaan paras mahdollinen testaustulos, kun integrointitestauksen hyväksymiskriteerit ja järjestelmätestauksen aloituskriteerit on määritelty. Aloituskriteereinä voi olla edellisen tason hyväksymiskriteereitä sekä esimerkiksi testausympäristön perustaminen ja testidatan kerääminen. (Craig & Jaskiel 2002, 127–129.)

Järjestelmätestauksessa koko järjestelmä testataan yhtenä kokonaisuutena, koska yksittäiset osat on jo testattu aikaisemmin. Testaus on luonteeltaan mustalaatikkotestausta. Testausmenetelmistä kerrotaan luvussa 4. Tässä vaiheessa voi tulla ilmi, että laitteisto- ja komponenttikokoonpano eivät toimikaan oletetusti yhdessä, suorituskyky on liian alhainen, virhetilanteiden

toipuminen on heikkoa tai ohjelmiston turvallisuudessa voi olla puutteita. Jäntti (2003, teoksessa Paakki 2000, 15) esittää järjestelmätestauksesta seuraavia muotoja:

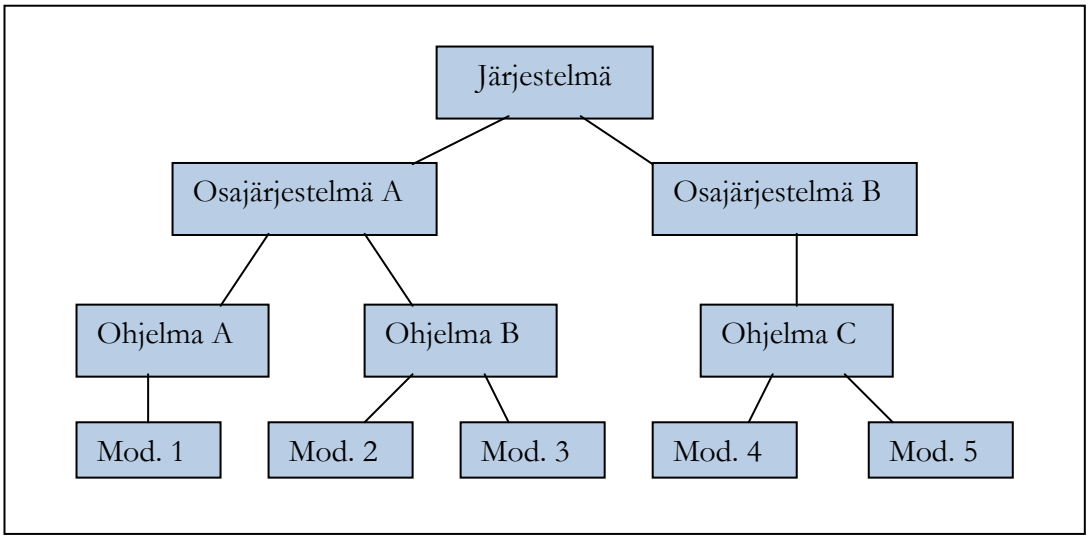
- volyymitestaus (volume testing) määrittelee, pystyykö järjestelmätestaus käsittelemään riittävää määrää tietoa tai pyyntöjä
- kuormitustestaus (load/stress testing) tunnistaa kuormitushuiput, joissa järjestelmä voi epäonnistua käsittelemään tietoa annetuilla aikarajoilla
- turvallisuustestaus (security testing) osoittaa, että järjestelmä täyttää sille asetetut turvallisuusvaatimukset
- suorituskykytestaus (performance testing) määrittelee, täyttääkö järjestelmä sille asetetut suorituskykyvaatimukset
- resurssien käytön testaus (resource usage testing) tarkistaa, käyttääkö järjestelmä liikaa resursseja kuten muistia tai levytilaa tms.
- konfiguraatiotestaus (configuration testing) näyttää, toimiiko järjestelmä oikein, kun ohjelmisto, laitteisto, tietokannat ja ulkoiset laitteet on liitetty yhteen
- asennettavuustestaus (installability testing) testaa, johtaako asennusprosessi epäonnistuneeseen asennukseen
- toipumistestaus (recovery testing) osoittaa, miten järjestelmä täyttää sille asetetut vaatimukset uudelleenkäynnistämiseen johtavan virheen tai muun virheen jälkeen
- luotettavuus/saatavuustestaus (reliability/availability testing) määrittelee, täyttääkö järjestelmä sille asetetut luotettavuus- ja saatavuusvaatimukset. Esim. kuinka usein järjestelmä on käytettävissä.

Integrointitestaus

Integrointitestauksessa yhdistellään yhteen moduuleita tai moduuliryhmiä (osajärjestelmiä). Testauksessa keskitytään moduulien välisten rajapintojen toimivuuteen ja tiedonvaihdon oikeellisuuteen. On huomioitava, että muuttujia on voitu määritellä globaalisti tiedostojen tai tietokantojen avulla. Integrointitestausta suunniteltaessa käytetään hyväksi arkkitehtuurisuunnittelun (teknisen määrittelyn) dokumentteja, määrittely- ja suunnitteludokumentteja sekä käyttöohjeita. Arkkitehtuurisuunnittelussa määritellään karkean tason ohjelmistomoduulit ja niiden väliset yhteydet. Testauksen suunnittelu voidaan aloittaa, kun nämä ovat valmiina. Integrointitestaus käyttää hyväkseen moduulitestauksen tuloksia. Haikalan ja Märijärven (2006, 290) mukaan integrointitestaus voi edetä rinnan moduulitestauksen kanssa.

Vaikka yksittäiset moduulit onkin jo testattu moduulitestauksessa, integrointitestauksessa löydetään mahdolliset ongelmakohdat, joissa nämä moduulit eivät yhdessä toimi oikein. Virheellisessä tilanteessa esimerkiksi tietoa voi hävitä moduulien rajapinnoilla tai moduulit eivät ole yhteensopivia. Virheitä voi myös syntyä, jos suunnitteluvaiheessa ei ole huomioitu, voiko kutsuttu moduuli saada virheellistä tietoa vai ei tai mitä arvoja virhetilanteissa saadaan. Testausmenetelmänä voidaan käyttää Big Bang-menetelmää, jossa testaus tehdään liittämällä kaikki moduulit kerralla yhteen. Muita testausmenetelmiä on testata järjestelmää inkrementaalisesti liittämällä moduulit yhteen kokoavasti bottom-up – periaatteella alemman tason moduuleista ylöspäin. Jäsentävässä top-down – periaatteessa etenemissuunta on päinvastainen. Inkrementaalisen testauksen hyötynä on virheiden helpompi paikallistaminen alimoduuleista. Testausmenetelmistä on kerrottu enemmän kappaleissa 4. (Jäntti 2003, 14.)

Integrointitestaus olisi hyvä aloittaa kriittisimmistä kohdista, joissa tn. tulee vastaan eniten ongelmia, jotta ne saadaan selvitetyksi. Testauksen laajuus vaihtelee kohteen mukaan. Testattavana voi olla yksittäinen ohjelma tai laaja järjestelmä, joka koostuu monista osajärjestelmistä, ohjelmista ja niiden moduuleista. Integrointitestauksen hierarkiaa on kuvattu kuviossa 7. Testiä suunniteltaessa tulisi määritellä mitkä moduulit tai kohteet testataan ryhmänä, mitkä ovat kriittisiä piirteitä, kuinka paljon testausta suoritetaan, milloin testaus aloitetaan ja mitkä ovat testauksen lopettamiskriteerit. (Craig & Jaskiel 2002, 130–135.)



Kuvio 7. Integrointitestauksen hierarkiaa

Moduulitestaus

Moduulitestauksessa testattavana on yksi moduuli, joka koostuu yleensä n. 100–1000 ohjelmarivistä. Testauksen suunnittelussa käytetään hyväksi moduulisuunnittelun ja edellisten tasojen määritelmiä. Suunnitteluvaiheessa laaditaan testitapausten luettelo ja suunnitellaan testaamistapa. Moduulin toimintaa verrataan moduulisuunnittelun ja arkkitehtuurisuunnittelussa lähinnä teknisen määrittelyn tuloksiin. Testauksen lähtökohtana on, että ohjelmointi on tehty. Testauksen suorittaa yleensä moduulin toteuttaja. Moduulin toteuttamiseen voidaan joutua laatimaan testiajureita (test drivers) ja tynkämoduuleita (test stubs). Testiajurit mahdollistavat moduulien toteuttamien palveluiden kutsumisen ja tulosten tarkastelun. Tynkämoduulit esittävät aliohjelmaa, jota testattava moduuli kutsuu. (Haikala & Märijärvi 2006, 289.)

Ohjelmarivin pieni määrä helpottaa moduulitestauksessa virheiden löytymistä ja tässä vaiheessa virheiden korjauskustannukset ovat vielä pieniä. Moduulitestaus on luonteeltaan lasilaatikkotestausta, ts. moduulien rakenne tunnetaan yksityiskohtaisesti. Kun moduulit on testattu, voidaan testauksen ylemmillä tasoilla luottaa yksittäisten moduulien toimivuuteen. Moduulitestauksessa tehokkaita virheiden tai huonosti toteutettujen ratkaisujen etsintäkeinoja ovat tarkastukset ja läpikäynnit. Eräs tehokas keino on myös parityöskentely, jossa henkilö A kirjoittaa testitapaukset määritysten pohjalta valmiiksi henkilölle B ennen koodin kirjoitusta ja päinvastoin. Testitapausten määrittäminen ennen koodauksen aloittamista helpottaa itse koodaustyötä. Parityöskentelyn toisena hyvänä puolena voidaan pitää tietotaidon jakamista useammalle henkilölle. (Craig & Jaskiel 2002, 140–143.)

Partasen (2009, 21) mukaan ohjelmistoprojektin luonteesta riippuen moduuleita voidaan ottaa integrointitestaukseen varhaisessa vaiheessa, jolloin moduuli sisältää vain perustoimintoja. Näin pystytään lisäämään projektin joustavuutta, mutta tämä edellyttää hyvää kommunikointia ohjelmistokehittäjien ja testaajien välillä.

4 Testausmenetelmät

Testausmenetelmän valinta riippuu testattavan ohjelmiston luonteesta, testaustasosta ja testaajien testaustaidoista. Menetelmät jaetaan staattisiin ja dynaamisiin testausmenetelmiin. Seuraavissa luvuissa kerrotaan enemmän näistä testausmenetelmistä ja testitapausten valinnasta.

4.1 Staattiset menetelmät

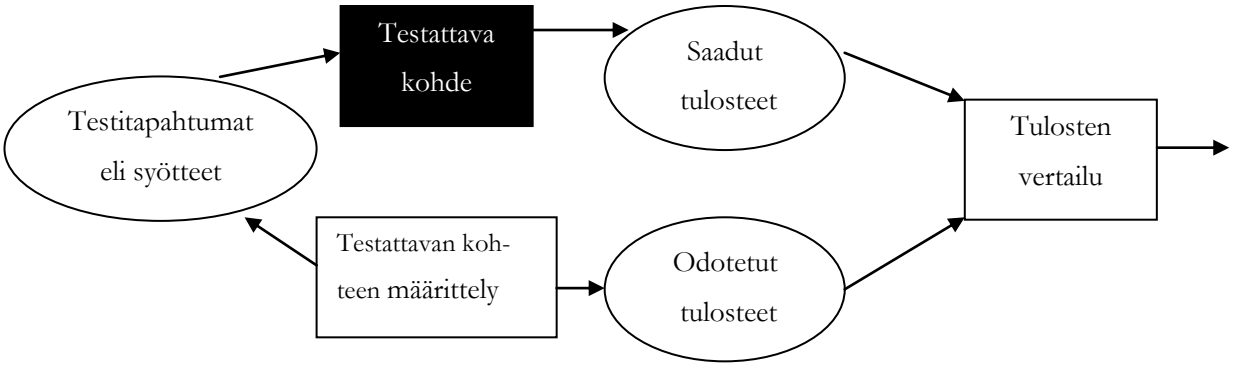
Ohjelmiston staattinen analyysi tarkoittaa ohjelmiston arkkitehtuuri- ja moduulisuunnitelmien tai itse ohjelmakoodin tarkastelua ja analysointia ilman ohjelmakoodin suorittamista. Analysointi voidaan suorittaa joko käsin tai automaattisesti koodin analysointiohjelmalla. Staattinen analyysi voi paljastaa muuttujiin tai parametreihin liittyvät virhekäytöt, indeksien ja osoittimien virheellisen käytön tai funktiokutsuihin liittyvät ongelmat, kuten kutsumattomat funktiot. (Partanen 2009, 29.) Staattisiin menetelmiin kuuluvat tarkastukset, katselmoinnit ja läpikäynnit, joista on puhuttu luvussa 2.4 Laadunvarmistus. Staattista analyysiä voidaan suorittaa eri testaustasoilla, mutta näiden painottuminen testauksen alkupäähän pienentää kustannuksia.

4.2 Dynaamiset menetelmät

Dynaaminen analyysi testausmenetelmänä on ohjelmiston varsinaista testaamista ohjelmakoodia suorittamalla niin, että pyritään löytämään virheitä. Tällöin keskitytään ohjelman ja koodin käyttäytymiseen ohjelman suorituksen aikana.

Mustalaatikkotestaus

Mustalaatikkotestaus (Black-Box testing) perustuu testattavana olevan kohteen (ohjelma tai funktio) syötteisiin ja tulosteisiin. Mustalaatikkotestauksessa ei välitetä testattavana olevan kohteen rakenteesta tai sisällöstä (koodista), vaan tutkittavana ovat kohteen tulosteet erilaisilla syötearvoilla. Testaajalle kohde on ikään kuin musta laatikko. Kuten kuviosta 8 nähdään, testauksen oikeellisuutta tarkastellaan vertaamalla saatuja tuloksia oikeisiin tai haluttuihin tulosteisiin. Ohjelman ja sen ympäristön pitää jäädä suorituksen jälkeen oikeaan tilaan. Testitapaukset johdetaan aina kohteen määrittelyn perusteella. (Enqvist ym. 2001, 8.)



Kuvio 8. Mustalaatikkotestaus (Enqvist ym. 2001, 8)

Testitapausten suunnittelu voidaan aloittaa, kun vaatimusmäärittely ja määrittelydokumentti on tehty, siis ennen kuin ohjelmakoodia on olemassa. Testitapausten suunnittelu jo tässä vaiheessa auttaa parantamaan moduulisuunnittelua ja ohjelmakoodin kirjoittamista. Testitapausten valintaan mustalaatikkotestauksessa on olemassa mm. seuraavia tekniikoita:

Ekvivalenssiositus (Equivalence partitioning)

Tässä tekniikassa on pyrkimyksenä jakaa testattava aineisto ekvivalenssiluokkiin siten, että testauksesta tulee mahdollisimman kattava ja testitapausten määrä pysyisi kuitenkin pienenä. Kaikilla mahdollisilla syötteillä on lähes mahdotonta testata ohjelmistoa tai se ei ole ainakaan järkevää. Ideana on jakaa testitapaukset siten, että yhteen ekvivalenssiluokkaan tulevat syötteet, joita ohjelmisto käsittelee samalla lailla ja tuottaa samanlaisia tulosteita. Mikä tahansa arvo luokassa edustaa koko luokkaa. Lisäksi valitaan vielä ekvivalenssiluokka, johon tulevat kelvottomat tai laittomat arvot. Kun jako on tehty, testitapaukset voidaan valita kattavasti eri ekvivalenssiluokista. Kuviossa 9 on havainnollistettu testitapausten jakoa. Esimerkkinä ottoautomaatti, josta voi nostaa 20 € seteleitä 20 - 200 €:n väliltä. Testitapaukset voidaan jakaa kolmeen eri ekvivalenssiluokkaan, joista jokaisesta otetaan yksi testitapaus testattavaksi. Kattavaan testaukseen riittää tässä tapauksessa kolme testitapausta. (Craig & Jaskiel 2002, 162–161.)

Kuvio 9. Esimerkki ekvivalenssiluokkiin jaosta (Craig & Jaskiel 2002, 162–161)

Raja-arvoanalyysi (Boundary value analysis)

Tässä tekniikassa testattavat tapahtumat valitaan nimensä mukaisesti rajoilla olevista äärimmäisistä arvoista. Jos käytetään ekvivalenssiluokitusta, tulevat raja-arvotkin usein käytännössä testattua. Raja-arvot ovat erittäin tärkeitä testauksen kohteita, koska niissä tapahtuu herkästi virheitä. Craigin ja Jaskielin (2002, 166) mukaan raja-arvojen lisäksi testataan arvo, joka on välittömästi ylimmän raja-arvon yläpuolella ja alimman raja-arvon alapuolella. Esimerkin mukaan, joka esiteltiin kuviossa 9, testattavia arvoja olisivat:

- 0 €, joka on alimman raja-arvon alapuolella (kelvoton)
- 20 € alin raja-arvo (kelvollinen)
- 200 € ylin raja-arvo (kelvollinen)
- 220 €, joka on ylimmän raja-arvon yläpuolella (kelvoton).

Jos syötejoukko on järjestetty, testataan ensimmäinen ja viimeinen syöte.

Virheen arvaus (Error guessing)

Enqvist (2001, 9) esittelee virheen arvauksen yhtenä paljon käytettynä testausmuotona. Menetelmä perustuu testaajan ammattitaitoon, kokemukseen, sovellusalueen tuntemukseen ja aikaisempiin testiraportteihin. Virheen arvaus-menetelmässä listataan kaikki mahdolliset mieleen tulevat testitapaukset, joilla testaus saadaan epäonnistumaan ja suoritetaan testaus näillä tapauksilla. Tyypillisesti 0 tai tyhjä ovat arvoja, jotka aiheuttavat ongelmia ohjelmistoissa. Tämä on hyvä lisä muille menetelmille, mutta ei itsessään täysin kattava.

Sattumanvarainen testaus (Random testing)

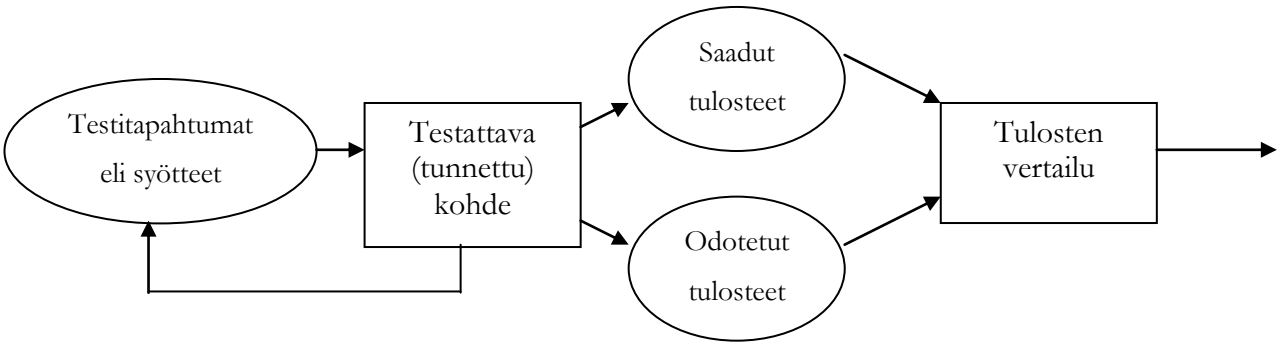
Tässä tekniikassa testiaineisto valitaan umpimähkään, yleensä jonkin työkalun avulla ja testaus suoritetaan tällä syötejoukolla. Tämä tekniikka ei ole suositeltava, koska sattumanvaraisesti valittu testiaineisto voi kuulua ainoastaan yhteen edellä mainittuun ekvivalenssiluokkaan. Testiaineistosta ei näin ollen saada luotettavasti kattavaa. (Craig & Jaskiel 2002, 175–176.)

Käyttöliittymän testaus

Jäntti (2003, 19) ottaa esille käyttöliittymän testauksen yhtenä mustalaatikkotestausmenetelmänä. Käyttöliittymää testattaessa ohjelmiston sisäinen toteutus on piilossa. Tätä testausmenetelmää käytetään järjestelmä- ja hyväksymistestaustasoilla. Käyttöliittymävirheitä voivat olla mm. näyttövirheet (tieto on väärässä paikassa) tai virheellinen suoritusjärjestys (tallennus jää kesken).

Lasilaatikkotestaus

Lasilaatikkotestauksessa (White-Box testing, Glass-Box testing) testaaja näkee ohjelmiston rakenteen eli koodin. Testitapahtumat voidaan suunnitella koodin perusteella. Kuviossa 10 on havainnollistettu lasilaatikkotestauksen toimintapataa. Testaus tapahtuu nimenomaan ohjelmiston toiminnallisuuden perusteella eikä testaaja välitä ohjelmistolle asetetuista vaatimuksista. Lasilaatikkotestausta suoritetaan moduulitestaustasolla, koska testauksessa keskitytään yksityiskohtiin. (Enqvist ym. 2001, 10.)



Kuvio 10. Lasilaatikkotestaus (Enqvist ym. 2001, 10)

Lasilaatikkotestauksessa testaus etenee loogisia polkuja pitkin. Tämän takia testitapausten valinnassa tavoitteena on, että kaikki kohteen (ohjelman tai funktion) haarat ja ohjelmapolut tulisivat käytyä läpi. Testidatan valinnalla pyritään mahdollisimman kattavaan testaukseen. Kattavuusmitoilla yritetään varmistua, että ohjelman kaikki osat tulisi kattavasti suoritettua. Testauksen kattavuutta voidaan mitata eri tavoilla ja nämä eri tavat määräävät myös testidatan valintaa. Korkein kattavuus saavutetaan moduulitestaustasolla. Edettäessä V-mallissa ylöspäin kattavuus yleensä laskee. Seuraavaksi esiteltyinä koodikattavuuteen liittyviä tekniikoita, joiden kriteerien mukaan testitapaukset voidaan valita: (Haikala & Märijärvi 2006, 294–296.)

Lausekattavuus (Statement coverage)

Lausekattavuudella tarkoitetaan, että ohjelman jokainen lause suoritetaan vähintään kerran. Todellisuudessa päästään harvoin tilanteeseen, jossa saavutetaan 100 % lausekattavuus. Haikalan ja Märijärven (2006, 295) mukaan pienenkin koodimoduulin todellinen kattavuus ylittää harvoin 90 %.

Polkujen kattavuus (Path coverage)

Tekniikan mukaan kaikki ohjelman polkujen vaihtoehdot tulisi pystyä suorittamaan valituilla testitapauksilla. Polulla tarkoitetaan ohjelman kohtaa, jossa on kaksi tai useampia vaihtoehtoisesti suoritettavia lauseita (if-else, switch-case jne.). Täydellinen polkujen kattavuus on mahdollista saavuttaa vain yhden yksinkertaisen funktion sisällä.

Ehtokattavuus (Condition coverage)

Ehtokattavuudessa edellytetään, että päätöksen kaikkien osaehtojen on saatava molemmat arvonsa (tosi/epätosi).

Päätöskattavuus (Decision coverage)

Tekniikassa edellytetään, että ehtorakenteen päätös saa vähintään kerran molemmat arvonsa (tosi/epätosi).

Harmaalaatikkotestaus

Harmaalaatikkotestaus (Gray box testing) on musta- ja lasilaatikkotestauksen välimuoto. Tekniikassa käytetään hyväksi tietoa ohjelman toteutusperiaatteista. Testataan yleensä raja-alueita kuten ali- ja ylivuotoja, nollalla jakamista, pyöristyksiä ja esim. kuukauden alku- ja loppupäiviä. Siirryttäessä V-mallissa alimmalta tasolta ylöspäin testauksen luonne muuttuu lasilaatikkotestauksesta enemmän mustalaatikkotestaukseksi. (Haikala & Märijärvi 2006, 291.)

Top-dowm-testaus

Top-down-tekniikassa testaus aloitetaan ohjelman korkeimman tason moduuleista ja tämän jälkeen edetään alemman tason moduulien ja komponenttien testaukseen. Koska testaus aloitetaan ylemmältä tasolta, täytyy alemman tason moduuleita simuloida. Tähän tarvitaan ”tyhmiä” moduuleita tai moduulin pätkiä (module stubs), jotka ovat yksinkertaisia eivätkä sisällä monimutkaista toiminnallisuutta. Kun testaus etenee alemmalle tasolle, stub-moduulit korvataan oikeilla moduuleilla. Etuina tässä menetelmässä ovat varhaisessa vaiheessa saatava alustava ohjelma sekä kahden mahdollisesti erillisen testitason yhdistyminen esim. integrointi- ja järjestelmätestitasojen. (Enqvist ym. 2001, 11–12.)

Bottom-up-testaus

Bottom-up-tekniikassa testaus aloitetaan alimman tason moduuleista, jotka eivät kutsu tai käytä muita moduuleita. Testaus etenee alhaalta ylöspäin, kunnes koko ohjelma on testattu. Tekniikka vaatii testiajureiden (test drivers) luomisen, jotta alimman tason moduuleita voidaan suorittaa ja testata. Tämän menetelmän etuna on testiaineiston helppo valinta. Jos moduulien alimmilla tasoilla on kriittisiä tai virhealttiita kohtia, kannattaa käyttää bottom-up-tekniikkaa. Haittapuoleksi voidaan katsoa, että mitään ohjelmaa ei ole valmiina ennen kuin viimeinenkin ylimmän tason moduuli on testattu. Testausta ei myöskään voida aloittaa ennen kuin alimmankin tason moduulit on suunniteltu ja määritelty. (Enqvist ym. 2001, 12.)

5 Testauksen riittävyyden arviointi

Milloin testaus sitten pitäisi lopettaa? Etukäteen on vaikea arvioida kuinka kauan testejä tulisi suorittaa. Testaussuunnitelmassa tulisi olla hyväksymiskriteerit, milloin testaus on hyväksytty ja voidaan siirtyä seuraavaan vaiheeseen. Kriteerinä voi olla esimerkiksi löydettyjen virheiden määrä suhteessa korjattuihin virheisiin. Kun virhekäyrä tasaantuu, voidaan testaus lopettaa. Apuna voidaan myös käyttää kattavuusmittoja, joista on kerrottu luvussa 4.2 lasilaatikkotestausmenetelmien yhteydessä. (Haikala & Märijärvi 2006, 293.) Mitä nämä virheet ovat, mitä testauksessa etsitään?

5.1 Virheet, viat ja häiriöt

Testauksessa keskitytään virheiden etsimiseen ja niiden eliminoimiseen. Testauksen yhteydessä virheet voidaan jaotella eri ryhmiin Haikalan ja Märijärven (2006, 287–288) mukaan seuraavasti:

- virhe (error) on ihmisen (ohjelmistokehittäjän) tekemä poikkeama spesifikaatiosta. Tästä seuraa, että ilman spesifikaatiota testaus on mahdotonta, koska lopputulosta ei voida todentaa
- vika (fault) voi aiheutua järjestelmään, kun tällainen virheellinen kohta on ajettu. Vika voi korjautua itsestään toisen toiminnon seurauksena, mutta se voi myös aiheuttaa järjestelmään häiriön
- häiriö (failure) näkyy järjestelmän ulkoisessa toiminnassa. Esimerkiksi ottoautomaatilta nostettaessa tilin saldo laskettaisiin väärin.

Termien käyttö ei ole vakiintunut. Yksi yleinen tulkinta on, että virhe (error) on ihmisen aiheuttama teko, jonka seurauksena syntyy vika (fault). Yleinen arvio virheiden määrästä on, että ohjelmoinnin jälkeen löytyy yksi virhe muutamaa kymmentä ohjelmariviä kohden. Pitempään käytössä olleista ohjelmista arvioidaan löytyvän yksi virhe tuhatta ohjelmariviä kohden.

Virheiden korjaus kannattaa keskittää sinne, mistä niitä eniten löytyy. Jäntti (2003, 26) on esittänyt Pro Gradu tutkielmassaan, että virheitä esiintyy eniten ohjelmien suorituslogiikassa kuten

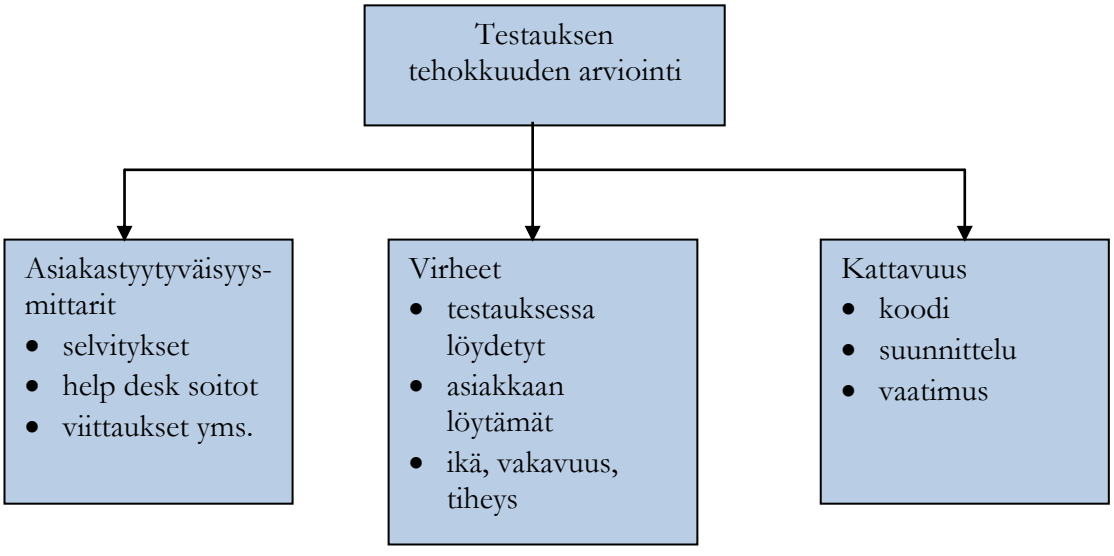
ehtolausekkeissa, toisto- ja silmukkarakenteissa. Rajapintavirheiden määrä on myös lisääntynyt komponenttipohjaisten sovellusten myötä.

5.2 Testauksen tilanneraportti

Yleistestaussuunnitelmassa kerrotaan, miten testauksen tilannetta seurataan. Testauksen tilanneraportissa kerrotaan, mitä testauksia on tehty: testausten määrä, löydettyjen virheiden sijainti ja luokitusaste (kuinka vakavasta virheestä on kyse) sekä testauksen kattavuus. Raportti voi olla taulukkomuodossa, josta tilannetta on helppo seurata. Tilanneraportin avulla voidaan yhteistyökumppaneille jakaa tietoa testauksen tilanteesta. Tällainen raportti kertoo testattavien testitapausten määrästä. Mittarina voidaan käyttää myös aikapohjaista mittausta, jossa kerrotaan testaukseen mennyt aika. Toiminnallisessa tilanteen määrittelyssä huomioidaan kuinka paljon testitapaukset kattavat toiminnallisuutta. Jotkin testitapaukset kattavat liiketoiminnalle tärkeitä piirteitä enemmän kuin toiset. (Craig & Jaskiel 2002, 258–259.)

5.3 Testauksen tehokkuuden arviointi

Kuinka tehokasta testaus on ollut? Näitä eri tekijöitä on yrityksissä Craigin ja Jaskielin (2002, 269–291) mukaan jaettu yleisesti kolmeen eri kategoriaan kuvion 11 mukaan. Monet näistä mittareista mittaavat testauksen tehokkuuden lisäksi myös itse tuotteen laatua.



Kuvio 11. Testauksen tehokkuuden arviointi (Craigin & Jaskielin 2002, 269)

Asiakastyytyväisyys-mittarit

Yleisesti käytettyjä asiakastyytyväisyys mittareita ovat soitot ohjelmistotukeen ja asiakastyytyväisyyskyselyt. Tyytyväisyyskyselyistä ei välttämättä saa objektiivista kuvaa todellisesta tilanteesta. Ongelmia voi tulla kysymysten asettelussa ja vastausten tulkinnassa. Vastauksista ei pystytä erottelemaan, mikä on ollut tehokkuutta testauksessa ja mikä on ollut itse ohjelmistoprojektin osuus. Asiakastyytyväisyyskyselyt antavat hyvän yleisen mielipiteen kokonaistilanteesta. Asiakastyytyväisyydellä mitataan kuitenkin ohjelmistoa vasta jälkeenpäin, kun se on jo tuotannossa. Ne antavat arvokasta tietoa yritykselle ja testaajille, mutta eivät pelkästään ratkaise ongelmaa, miten testauksen tehokkuutta mitataan.

Virheet ja niiden mittaaminen

Testauksessa löydetyt virheet kannattaa jaotella niiden vakavuuden mukaan, jotta saataisiin todenmukaisempi kuva testauksesta. Luonnollisesti, jos ohjelmistossa on paljon virheitä, löytyy myös testauksessa paljon virheitä. Eli tämä mittaa myös itse tuotteen laatua. Yleisempi mittari on löydetyt virheet tuotannossa tai asiakkaan toimesta. Näissäkin mitataan tehokkuutta, kun ohjelmisto on jo tuotannossa. Näistä mittareista onkin hyötyä yritykselle, kun katsotaan testausten tehokkuuden kehitystä pitkällä aikavälillä, ei niinkään kyseisen projektin kohdalla. Tähän ryhmään kuuluu myös virheen ikä, jossa virhe on sitä vanhempi, mitä myöhemmin se löydetään. Kuten aikaisemmin on jo todettu, virheen löytäminen ja korjaaminen tulee sitä kalliimmaksi, mitä myöhemmin se löydetään.

Virheiden tiheys lasketaan:

$$\frac{\text{löydetyt virheet}}{\text{ohjelmakoodin rivimäärä}}$$

Yleensä, jos jossain testattavassa ohjelmakohdassa virhetiheys on ollut suuri, virheitä löytyy jatkossakin. Virhetiheyden laskeminen auttaa testaajia jatkossakin keskittymään paremmin osa-alueisiin, joissa virhetiheys on ollut suuri.

Kattavuus

Kattavuusmitat mittaavat tehokkaimmin testauksen tehokkuutta, koska näitä mittareita käytetään jo testauksen aikana, eikä vasta tuotantoon oton jälkeen. Vaatimusten kattavuus voidaan mitata testitapausten suunnittelun jälkeen, ennen kuin koodia on kirjoitettu tai kun itse testaus suoritetaan. Suunnittelun kattavuuden mittaamista tarvitaan eri ehtojen läpikäymiseksi, joita vaatimuksissa ei vielä ole. Koodikattavuutta pidetään tärkeimpänä kattavuuden mittana. Kattavuuden mittaamiseen, jolla mitataan lause- ja polkukattavuutta, on olemassa työkaluja. Työkaluja käytettäessä koodikattavuuden mittarit kertovat, että ohjelmakoodia on testattu, mutta ne eivät kerro täyttääkö ohjelma sille alun perin asetetut vaatimukset. Siksi onkin tärkeää, että testitapauksia suunnitellaan vaatimusten, suunnittelun ja koodin perusteella. Näin voidaan taata, että vaatimuksiin kirjatut asiat ovat myös koodissa. Koodikattavuus on tehokasta etenkin moduuli- ja integrointitestaustasoilla käytettäessä.

6 Testaussuunnitelma

Testaussuunnitelma voi olla erillinen dokumentti tai se voi kuulua osaksi projektisuunnitelmaa. IEEE standardin 829-1998 (IEEE standardi 829-1998) mukaan testauksesta löytyvät seuraavat tasot: yksikkö/moduuli, integrointi, järjestelmä ja hyväksyminen. Projektin suuruudesta riippuu tarvitaanko eri tasoille oma testaussuunnitelma vai riittääkö yksi yleinen testaussuunnitelma. Testaussuunnitelman yleiskuvaus voi olla osana projektisuunnitelmaa. Järjestelmätestaussuunnitelma voi olla osa toiminnallista määrittelyä. Integrointi- ja moduuli (yksikkö) testaussuunnitelma voi olla osa teknistä määrittelyä. Testaussuunnitelmaa voidaan pitää kommunikoinnin välineenä muihin ohjelmistoprojektin jäseniin päin.

6.1 Testaussuunnitelman tarkoitus

Testaussuunnitelma on lähtökohta hyvälle ohjelmistotestaukselle. Testaussuunnitelman tarkoituksena on löytää kaikki ne tekijät, joita ohjelmistoprojektissa halutaan testata. Suunnitelma vastaa kysymyksiin mitä, miten ja milloin sekä millaisia lopputuloksia odotetaan. Testaussuunnitelman tarkoitus on järjestää testaus eri vaiheisiin. Jokaisessa vaiheessa kerrotaan tavoitteet, jotka testauksella halutaan saavuttaa sekä testauksen aikataulu ja resurssit sekä mahdolliset rajoitteet. Päämääränä on pikemminkin käsitellä testausstrategiaan liittyviä tärkeitä kysymyksiä kuten resurssien käyttöä, vastuukysymyksiä, riskejä sekä painopistealueita kuin luoda pitkiä testauslistoja. Testaussuunnitelmaa tehtäessä täytyy myös ottaa huomioon kuka sitä lukee. Jos testaussuunnitelma tehdään jokaiselle vaiheelle erikseen, voidaan olla varmoja, että moduulitestaussuunnitelma on tarkoitettu eri henkilöille kuin esimerkiksi hyväksymistestaussuunnitelma. (Craig & Jaskiel 2002, 54–57.)

Testauksen suunnittelu sisältää testitapausten suunnittelun, testausmenetelmän valinnan ja testiaineistojen laadinnan. Testitapausten suunnittelussa identifioidaan jokin asia, esim. miten ohjelmisto toimii, kun sille antaa väärän syötteen. Määritellään, miten lähtötilanne saadaan aikaan ja mikä ohjelmiston oikea tila on tässä tapauksessa. (Rajamäki 2000, 12.)

6.2 Testaussuunnitelmat vaiheiden mukaisesti

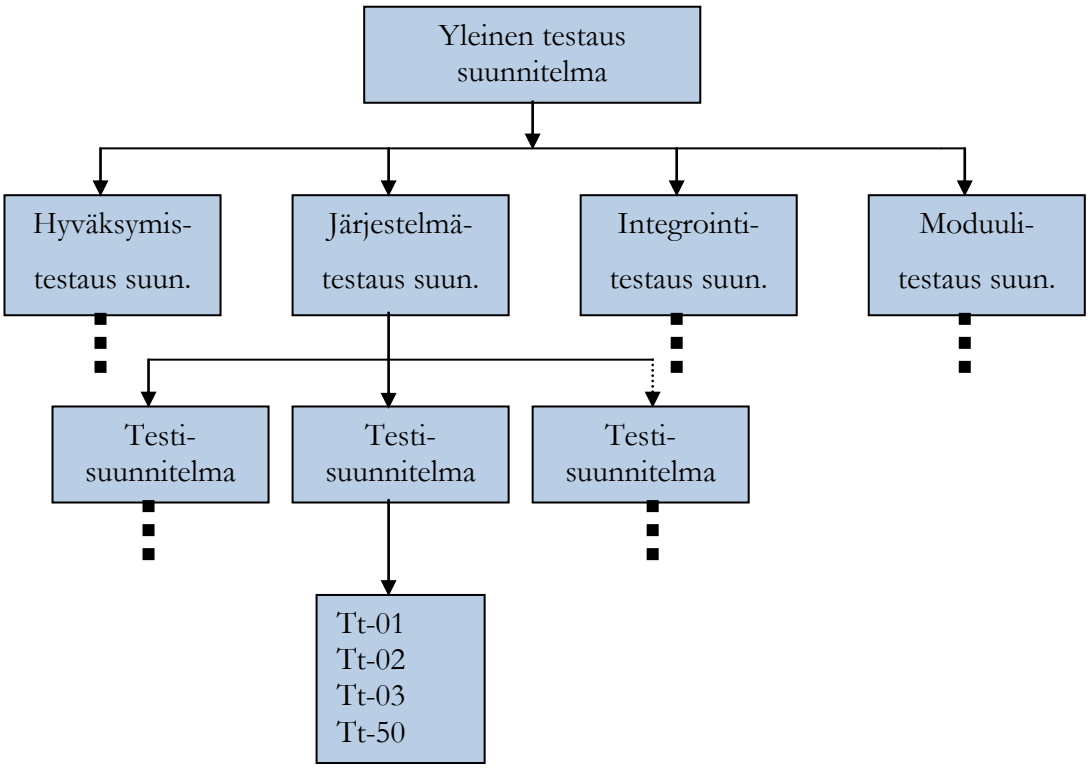
Jotta testauksesta tulisi mahdollisimman tehokasta, tulisi testauksen suunnittelu aloittaa yhtä aikaa sovelluskehityksen kanssa. Projektin yleisiä tietoja käytetään suunniteltaessa yleistä testaussuunnitelmaa, kun taas kehitettävän ohjelmiston yksityiskohtaisempaa tietoa käytetään suunniteltaessa yksityiskohtaisempia testaussuunnitelmia. Jokaisessa testaustasossa määritellään laitteisto, ohjelmisto, rajapinnat, data ja henkilöstö. Kehitettävän ohjelman laajuudesta riippuen tasoja voi olla vain yksi tai käyttämämme mallin mukaan neljä: hyväksymis-, järjestelmä-, integrointi ja moduuli/yksikkötestaustasot. Erilaisia testauskäytäntöjä on käytössä enemmänkin. Edellisten lisäksi puhutaan mm. alfa- ja betatestauksista, joita käytetään hyväksymistestauksen yhteydessä ja käytettävyystestauksesta, jota käytetään järjestelmätestauksen yhteydessä. Lisäksi käytössä on myös regressio- eli uusintatestaus, jota voidaan suorittaa kaikissa testaustasoissa. Tärkeintä määriteltäessä kuinka montaa eri tasoa käytetään, on määritellä kyseisen tason laajuus, mitä tason tulee toteuttaa ja suunnitella taso niin, että tämä toteutuu. (Craig & Jaskiel 2002, 98–100.)

Alfatestauksessa testaus suoritetaan toimittajan tiloissa asiakkaan toimesta, kun taas **betatestauksessa** testaus suoritetaan asiakkaan omissa tiloissa oikeassa suoritusympäristössä. **Regressiotestauksessa** on kyse uudelleentestauksesta. Kun virheitä huomataan esim. järjestelmätestauksessa ja havaittu virhe korjataan, voi tämä aiheuttaa muutoksia alemman tason moduuleihin. Tämä puolestaan aiheuttaa uusintatestejä jo testatuissa osissa. Lisäksi täytyy huomioida, että virheiden korjaus voi aiheuttaa uusia virheitä. Mitä ylemmällä tasolla regressiotestauksessa ollaan, sen kalliimmaksi vihreiden korjaus tulee. (Haikala & Märijärvi 2006, 290–291.)

Käytettävyystestauksen tarkoituksena on parantaa ohjelmiston käytettävyyttä. Tuotteen lisäksi käytettävyys muodostuu käyttäjästä, käyttöympäristöstä, suoritettavasta tehtävästä ja ohjelmiston ympärillä olevasta tekniikasta. Testauksen tulokset analysoidaan ja niiden pohjalta suoritetaan kehitys- tai korjaustoimenpiteet. Testaus aloitetaan määrittelemällä tarkkaan, mitä testauksesta halutaan saada selville, esimerkiksi käyttäjän tulee selvitä lehtitilauksen tekemisestä viidessä minuutissa. Testauksen kohteeksi valitaan toimintoja, joista oletetaan löytyvän käytettävyysongelmia. Käytettävyystestaus on yleensä käyttöliittymätestausta. (Jäntti 2003, 15–16.)

7 Testauksen suunnittelun dokumentit

Jotta testauksessa saavutetaan halutut ja asetetut tavoitteet ja jotta saavutettuja tuloksia pystytään seuraamaan, tulee testauksesta laatia dokumentteja. Dokumenteista on hyötyä myös, kun suunnitellaan uusia testauksia. Testituloksista saadaan tietoa, missä vaiheessa ohjelmistotestaus on menossa ja ongelmien sattuessa voidaan voimavaroja kohdistaa oikein. Testitulokset määrittelevät myös osaltaan ohjelmiston laatutasoa ja niiden avulla pystytään ilmoittamaan testauksen kattavuus. Mutta ilman hyviä testaussuunnitelmia ei saada aikaan testituloksia. Craig ja Jaskiel (2002, 188) ovat esitelleet kuvion 12 mukaisesti testaussuunnitelman hierarkian.



Kuvio 12. Testaussuunnitelman hierarkia (Craig & Jaskiel 2002, 188)

Craig ja Jaskiel (2002, 461) ovat esitelleet IEEE Standardin 829-1998 mukaiset testauksen dokumenttimallit:

1) Testaussuunnitelma (Master Test Plan / Test Plan)
- käytetään yleistestaussuunnitelman ja eri testaustasojen testaussuunnitelmaa laadittaessa.

2) Testisuunnitelma (Test Design Specification)
- käytetään testitapausten määrittämiseen, tässä kerätään yhteen samanlaiset testitapaukset
- määritellään tarpeelliset testitapaukset eri piirteiden testaamiseen.

3) Testitapausten määrittely (Test Case Spesification)
- käytetään tarvittaessa yksittäisten testitapausten määrittämiseen, testien syötteet ja odotetut tulokset.

4) Testausmenettely (Test Procedure Specification)
- määritellään testin suorittamisen vaiheet.

5) Testiloki (Test Log)
- käytetään tarvittaessa testauksen teknisenä kirjanpitona.

6) Testauksen tapahtumaraportti (Test Incident Report)
- raportti, jossa kerrotaan mitä testissä todella tapahtui ja virheistä tarkempi seloste.

7) Testauksen yhteenvetoraportti (Test Summary Report)
- raportti, jossa kerrotaan aikaansaannokset ja arviointi sekä resurssien käyttö.

Standardi pitää sisällään kattavan määrän dokumentteja. Riippuen täysin siitä onko testattavana kokonaan uusi järjestelmä vai onko kyseessä ohjelmiston päivitysversio, käytettävien dokumenttien määrä vaihtelee. Pienemmissä projekteissa voi dokumentteja myös yhdistellä keskenään. Esimerkiksi testisuunnitelma voi pitää sisällään myös testitapausten määrittelyn ja testausmenettelyn. Seuraavaksi enemmän testauksen suunnitteluun liittyvistä dokumenteista kohdista 1 – 4, mitä ne pitävät sisällään. Testauksen tuloksiin liittyvistä dokumenteista kerrotaan luvussa 8.

7.1 Testaussuunnitelma

On tärkeää, että yrityksellä on käytössään yhtenäinen malli testaussuunnitelmasta. Näin projektiin osallistuvat henkilöt sekä projektista kiinnostuneet pystyvät seuraamaan itse projektin ja testausprosessin etenemistä. Taulukossa 1 on IEEE standardin 829-1998 mukainen testaus-

suunnitelman mallipohja, josta on helppo muokata omien tarpeiden mukainen. Projektista riippuen, kaikkia mallipohjan kohtia ei välttämättä tarvita ollenkaan. Craig ja Jaskiel (2002, 60–61) ovat muokanneet hieman standardin mallia. Heidän mielestään mm. riskit tulisi jakaa kahteen osaan: ohjelmistoriskit sekä suunnitellut ja suunnittelemattomat riskit. Lisäksi he ovat lisänneet kolme kohtaa: sisällysluettelon (table of contents), viittaukset (references) ja sanaston (glossary). Oheinen mallipohja on tehty heidän mallipohjansa mukaan. Mallipohjaa voidaan käyttää suunniteltaessa minkä tahansa tason testaussuunnitelmaa (yksikkö-, integrointi-, järjestelmä-, hyväksymis- tai yleinen testaussuunnitelma.)

Testaussuunnitelmassa (Master Test Plan / Test Plan) otetaan kantaa

- ajankohtaan, milloin testaajat tulevat mukaan projektiin ja kuinka paljon heitä tarvitaan
- testauksen aloittamisajankohtaan
- pilottiversion käyttöön, jos kysymyksessä uusi ohjelmisto
- testaustekniikoihin
- testaustasojen käyttöön (hyväksymis, järjestelmä, integrointi, yksikkö/moduuli).

Taulukko 1. IEEE Standardin mukainen testaussuunnitelman mallipohja (Craig & Jaskiel 2002, 62)

NRO	CONTENTS	SISÄLTÖ
1.	Test Plan Identifier	Testaussuunnitelman tunnus
2.	Table of Contents	Sisällysluettelo
3.	References	Viittaukset
4.	Glossary	Sanasto
5.	Introduction	Johdanto
6.	Test Items	Testattavat kohteet
7.	Software Risk Issues	Ohjelmistoriskit
8.	Features to Be Tested	Testattavat ominaisuudet
9.	Features Not to Be Tested	Ominaisuudet, joita ei testata
10.	Approach	Strategia / lähestymistapa
11.	Item Pass/Fail Criteria	Läpäisy- ja hylkäyskriteerit
12.	Suspension Criteria and Resumption Requirements	Keskeyttämisen ja jatkamisen perusteet
13.	Test Deliverables	Testauksen tulokset

14.	Testing Tasks	Testaukseen liittyvät tehtävät
15.	Environmental Needs	Testausympäristö
16.	Responsibilities	Vastuut
17.	Staffing and Training Needs	Henkilöstö- ja koulutustarpeet
18.	Schedule	Aikataulu
19.	Planning Risks and Contingencies	Suunnitellut riskit ja niiden hallinta
20.	Approvals	Hyväksyjä

Seuraavaksi hieman tarkemmin, mitä mallipohjan kohdat pitävät sisällään Craigin ja Jaskielin (2002, 62–95) mukaan.

1) Testaussuunnitelman tunnus

Testaussuunnitelman yksilöivä tunnus, josta nähdään mikä versio on kyseessä, testaustaso sekä mihin ohjelmistoprojektiin se kuuluu.

2) Sisällysluettelo

Sisällysluettelosta näkyy testaussuunnitelman aiheet, viittaukset, sanasto ja viitteet. Tämä ei kuulu IEEE 829-1998 standardiin.

3) Viittaukset

Viittauksiin voi kuulua mm. projektivaltuutus, projektisuunnitelma, laadunvarmistussuunnitelma, vaatimusmäärityskuvasto tai muu asiaankuuluva dokumentti. Viittauksista tulee käydä ilmi dokumentin nimi, päiväys ja mistä se löytyy. Tämä ei kuulu IEEE 829-1998 standardiin.

4) Sanasto

Sanastossa määritellään kaikki termit ja lyhenteet. Täytyy myös muistaa kenelle testaussuunnitelma on laadittu. Sanasto tuottaa lukijalle lisäinformaatiota. Tämä ei kuulu IEEE 829-1998 standardiin.

5) Johdanto

Johdanto käsittää kaksi pääasiaa: perustiedot projektin tai päivitettävän version laajuudesta sisältäen avainpiirteet ja lyhyen historian, sekä testaussuunnitelman laajuuden. Testaussuunni-

telmasta voidaan esimerkiksi kertoa, mitkä testaustasot suunnitelma kattaa. Suunnitelma voi kattaa koko tuotteen testauksen, joka pitää sisällään myös laitteiston tai vain ohjelmiston testaamisen. Voidaan suorittaa vain pelkkä testaus tai mukana voi olla muitakin tekniikoita, kuten katselmukset, läpikäynnit ja tarkastukset.

6) Testattavat kohteet

Tässä osassa kuvataan, mitä testataan ja mitä saatetaan päätökseen yhteistyössä suunnittelijoiden kanssa. Tämä osa voi olla erilainen, riippuen testaussuunnitelman tasosta. Ylemmän tason suunnitelmassa tämä voi olla sovelluksittain järjestetty. Alemman tason suunnitelmissa tämä voi olla järjestetty ohjelma- tai moduulitasolla. Yleistestaussuunnitelmassa tässä voi olla esimerkiksi viittaus ohjelmiston vaatimusmäärittelyyn. Moduulitestaussuunnitelmassa tässä voi olla listattuna ohjelmat, joita testataan. IEEE standardin mukaan tulisi olla viittaukset, mikäli ne ovat olemassa

- vaatimusmäärittelyyn
- suunnittelun vaatimuksiin
- käyttöohjeisiin
- asennusohjeisiin
- tapauskohtaisiin raportteihin.

Jos jotain on jätetty tarkoituksella pois, se pitää mainita erikseen.

7) Ohjelmistoriskit

Riskien kartoituksella on tarkoitus määrittää mihin testauksessa keskitytään. Riskien kartoitus auttaa testaajia keskittymään testauksessa olennaisiin asioihin esim. kohtiin, joista voi löytyä virheitä tai jotka ovat tärkeitä liitetoiminnan kannalta. Ohessa esimerkkejä ohjelmistoriskeistä:

- rajapinnat toisiin järjestelmiin
- piirteet, jotka käsittelevät suuria rahasummia
- piirteet, joissa käsitellään suurta määrää asiakkaita
- monimutkaiset ohjelmat
- moduulit, joista on löytynyt aikaisemmin virheitä
- moduulit, joihin on tapahtunut paljon tai monimutkaisia muutoksia
- turvallisuus, suorituskyky ja luotettavuus asiat
- piirteet, joita on vaikea testata tai muuttaa.

Riskit on tässä mallipohjassa jaettu kahteen osaan: ohjelmistoriskit sekä suunnitellut riskit ja niiden hallinta, kohta 19.

8) Testattavat ominaisuudet

Tähän listataan kaikki testattavat asiat/toiminnot. Esimerkiksi pankkiautomaattia testattaessa testattaisiin mm. rahan nosto, tilin saldokysely ja maksutapahtuma. Alemmilla tasoilla testattavat piirteet voivat olla huomattavasti yksityiskohtaisempia. Testattavat piirteet voi priorisoida ohjelmistoriskien perusteella. Testattaville piirteille voidaan määritellä millä todennäköisyydellä riski toteutuu ja jos näin käy, mikä on sen vaikutus. Priorisointi auttaa määrittämään testausjärjestyksen ja jos aika käy vähiin, määrittämään matalan riskin piirteet, jotka voidaan jättää testaamatta ja siirretään kohtaan 9 piirteet, joita ei testata.

9) Ominaisuudet, joita ei testata

Tähän listataan ominaisuudet, joita ei testata ja miksi ei. Näitä voivat olla matalan riskin ominaisuudet sekä ominaisuudet, joita ei ole muutettu tai jotka eivät vielä ole käytössä.

10) Strategia / lähestymistapa

Tämä on testaussuunnitelman sydän. Tässä osassa kerrotaan, miten testaus tullaan toteuttamaan eli testausmenetelmät ja tuodaan esille ne asiat, joilla on vaikutusta testauksen onnistumiseen ja sitä kautta myös itse projektin onnistumiseen. Tällaisia asioita ovat mm.:

- resurssit kuten raha, aika, taidot ja henkilöstö
- testauksen tavoitteet kuten tarkastettavat rajapinnat tai tavoitteet joihin voidaan käyttää erilaisia mittareita, kuten esimerkiksi kuinka suuri osa suunnittelusta on testattu
- katselmustapa, kuten läpikäynnit ja tarkastukset
- testaustulosten vaatimukset, kuten täydellisyysaste, tarkkuustaso ja kattavuus
- testiympäristö
- riskeistä virheiden vaikutukset ja niiden todennäköisyys
- testaustyökalut, joita käytetään tai voidaanko jotain automatisoida
- ei-toiminnallisten vaatimusten testaaminen.

Jos testaussuunnitelma tehdään jokaiselle tasolle erikseen, tässä voidaan mainita niiden omat ominaispiirteet, kuten esimerkiksi ohjelmisto- ja laitteistokokoonpano ja ohjelmistojen väliset rajapinnat. Siinä tulisi myös mainita kriteerit, jolloin voidaan siirtyä tasolta toiselle. Mitä korke-

ammalla testaussuunnitelman tasolla ollaan, sitä realistisempi ympäristö on. Hyväksymistestauksessa esimerkiksi ympäristön tulisi vastata mahdollisimman paljon todellista ympäristöä.

Tässä kohdassa kerrotaan myös mitä tilastoa kerätään. Niiden avulla mitataan mm. testauksen tasoa, kattavuutta, tehokkuutta, ohjelmiston laatua ja aikataulussa pysymistä tms. Myös testaussuunnitelmaan tehtävistä muutoksista on hyvä mainita, kuinka usein suunnitelmaa päivitetään. Jos pidetään kokouksia, tässä on hyvä mainita niistä.

11) Läpäisy- ja hylkäyskriteerit
Tässä osiossa kerrotaan läpäisy-/hylkäyskriteerit kaikille testattaville kohteille, jotka on listattu kohdassa 6.

12) Keskeyttämisen ja jatkamisen perusteet
Tässä mainitaan kriteerit, milloin testaus keskeytetään tai milloin sitä voidaan jatkaa. Jos esimerkiksi havaitaan, että vakavia virheitä on löytynyt tietty määrä, testaus voidaan keskeyttää.

13) Testauksen tulokset
Tässä mainitaan kaikki tulosteet, joita testauksesta tuotetaan sekä työkalut tai vastaavat, joita kehitetään tai ylläpidetään testauksen aikana. Tulosteisiin kuuluvat esim. testaussuunnitelma, testitapaukset, virheraportit ja yhteenvetoraportti. Mainittakoon, että ohjelmisto ei kuulu tähän, vaan kohtaan 6 Testattavat kohteet.

14) Testaukseen liittyvät tehtävät
Tässä luetellaan tehtävät, joita tarvitaan itse testausta varten, kuten esim. jos tarvitaan erikoistaitoja.

15) Testausympäristö
Testausympäristöön kuuluvat laitteisto-, ohjelmistoympäristö, rajapinnat, testi data, palvelut, työkalut, julkaisut ja turvallisuustekijät tai mikä tahansa muu, jolla on vaikutusta testausympäristöön.

16) Vastuut

Vastuualueet on hyvä nimetä jokaiseen tehtävään erikseen. Tämä selkeyttää työn etenemistä ja eri osa-alueita on helpompi valvoa. Vastuualueita ovat esim. testauksen hallinta, suunnittelu, suoritus, valvonta ja testiympäristön valmistelu.

17) Henkilöstö- ja koulutustarpeet

Kartoitetaan, kuinka paljon henkilöstöä tarvitaan ja mitä taitoja heillä on oltava tai mitä koulutusta testausta varten tarvitaan esimerkiksi työkalujen käyttöön.

18) Aikataulu

Aikataulussa käy ilmi ajankohdat, milloin eri osa-alueet tai dokumentit on oltava valmiina. Riippuen testaustasosta, jonka suunnitelmaa ollaan tekemässä, aikataulutettavat asiat vaihtelevat. Yleistestaussuunnitelmassa aikataulutetaan lähinnä katselmukset, kun taas moduulitestaussuunnitelmassa aikataulutetaan yksittäisten moduulien testaus. Aikataulu voidaan myös antaa esim. tunteina tai päivinä, kuinka kauan jokin vaihe vie aikaa. Aikataulusta on myös hyvä tehdä piirros, josta näkee riippuvuudet muihin osa-alueisiin.

19) Suunnitellut riskit ja niiden hallita

Riskien kartoitus on hyvä tehdä suunnitteluvaiheessa. Jos riskeihin ei ole varauduttu etukäteen ja jotain odottamatonta tapahtuu, kuluu turhaan aikaa miettiä siinä vaiheessa mitä pitäisi tehdä. Kun riskejä tiedostetaan jo etukäteen ja tehdään toimintasuunnitelma valmiiksi, nopeuttaa tämä ongelmasta selviämistä. Tällaisia riskejä voivat olla esim. epärealistinen aikataulu, henkilöstöresurssit, budjetti, koulutustarpeet, huonosti suunniteltu ohjelmisto ja testauksen laajuus. Näihin voidaan valmistautua mm. aikataulua siirtämällä, henkilöstöä lisäämällä tai pienentämällä kyseisen ohjelmiston laajuutta.

20) Hyväksyjä

Hyväksyjä on henkilö, jolla on valtuudet hyväksyä kyseinen testausvaihe ja näin voidaan projektissa siirtyä seuraavaan vaiheeseen.

7.2 Testisuunnitelma

Yleisessä testaussuunnitelmassa määritellään testauksen strategia, aikataulut, riskit, vastuut jne., mutta siinä ei määritellä testitapauksia. IEEE standardin mukaan testitapaukset määritellään erillisellä testisuunnitelmalla (Test Design Spesification). Päämääränä on kerätä yhteen ryhmään samanlaiset testitapaukset. Näin joka tasolle (paitsi moduulitestaustasolle) syntyy yksi tai useampia dokumentteja. Testisuunnitelmassa käy kattavasti ilmi testitapaukset, mutta siinä ei kerrota miten ne toteutetaan. Tämä kerrotaan testausmenettelyssä (Test Procedure Specification). IEEE Standardin 829-1998 mukainen malli testisuunnitelmalle: (Craig & Jaskiel 2002,183–187.)

1) Mallin yksilöivä tunnus (Test Design Specification Identifier)
Tunnuksen avulla erotetaan testitapaukset toisistaan ja sillä viitataan siihen liittyvään testaussuunnitelmaan. Tunnisteet auttavat testitapausten hallinnassa.

2) Testattavat ominaisuudet (Feature(s) to Be Tested)
Kuvaus testitapauksista, joiden avulla haluttuja ominaisuuksia testataan. Kaikki testattavat ominaisuudet, jotka on mainittu testaussuunnitelmassa, täytyy löytyä testisuunnitelmasta.

3) Lähestymistapa (Approach Refinement)
Edetään testaussuunnitelman strategian mukaisesti, mutta yleensä tässä vaiheessa määritykset ovat yksityiskohtaisempia. Esimerkiksi jokin toiminto toimii vain tietyn ajan päivästä tai tiettyinä viikonpäivinä.

4) Testauksen tunnus (Test Identification)
Testitapaukset identifioidaan ja kerrotaan lyhyt kuvaus testitapauksesta. Tässä vaiheessa ei tarvitse kertoa testitapauksen yksityiskohtia, jos laaditaan erillinen raportti testitapausten määrittelemiseksi. Ottoautomaatti esimerkkitapauksessa testitapauksia olisivat:

- Tt01 – Nosta 20 € tililtä, jossa on rahaa 200 €
- Tt02 – Nosta 200 € tililtä, jossa on rahaa 200 €
- Tt03 – Nosta 200 € tililtä, jossa on rahaa 100 €.

5) Hylkäys- ja hyväksymiskriteerit (Feature Pass/Fail Criteria)

Määritellään kriteerit, milloin testi hylätään tai hyväksytään. Kriteerinä voidaan käyttää esim. että tietyn prosenttimäärän testitapauksista täytyy läpäistä testauksessa tai kaikkien testitapausten täytyy läpäistä testi. Kriteerinä voi myös olla esim. löydettyjen virheiden jakauma. Kun löydettyjen virheiden määrä pienenee, testaus voidaan lopettaa. Kriteerit täytyy määritellä testitapauksille erikseen ja niissä täytyy ottaa huomioon kuinka kriittisestä kohteesta on kysymys.

7.3 Testitapausten määrittely

Testitapausten määrittelyssä (Test Case Spesification) kuvataan yksittäiset testitapaukset. Mitä testataan ja minkälaisin tuloksin. Testitapaukset voidaan myös generoida automaattisilla työkaluilla. Suunnitteluvaiheessa ei yleensä saada kattavasti kerättyä kaikkia testitapauksia. Testauksen alettua testitapausten määrä kasvaa helposti ja niitä voidaan lisätä myöhemmin dokumenttiin. Seuraavaksi IEEE standardin mukainen testitapausten määrittelymalli: (Craig & Jaskiel 2002, 187–191.)

1) Testitapauksen yksilöivä tunnus (Test Case Specification Identifier)

Yksilöi testitapauksen. Yksilöivissä tunnuksissa voidaan käyttää päivämäärää, numeroa ja versiointia. Tämä helpottaa kun määritelmiin tehdään muutoksia.

2) Testauksen kohteet (Test Items)

Tässä kerrotaan kaikki tarvittavat asiat, joita testitapauksen testaamiseen tarvitaan. Näitä ovat esimerkiksi vaatimusmäärittelyn dokumentit, suunnittelun dokumentit ja itse koodi.

3) Syötteet (Input Specification)

Tässä kerrotaan mitä syötteitä testiä varten tarvitaan. Syöte voi olla esimerkiksi tiedosto, rajapinta toiseen systeemiin tai jokin arvoväliluokka, joka syötetään kenttään.

4) Tulosteet (Output Specification)

Tässä kerrotaan mikä on testauksen oletettu tulos. Oletettu tulos voi olla virheilmoitus, tietojen onnistunut tallennus, tiedon uusi arvo tms.

5) Laitteistovaatimukset (Environmental Needs)

Tässä kerrotaan mitä laitteistovaatimuksia testin suorittamiseksi tarvitaan.

6) Erikoisvaatimukset (Any Procedural Requirements)

Tässä kerrotaan, tarvitseeko testaus jotain erityisvaatimuksia ennen suoritusta tai täytyykö jonkin ehdon olla voimassa, jotta testaus voidaan suorittaa. Voidaan käyttää myös termiä esiehto. Ottoautomaatti esimerkkiä käyttäen, tilillä täytyy olla 200 €, jotta noston voi suorittaa.

7) Testitapausten väliset riippuvaisuudet (Inter-Case Dependencies)

Testitapaukset täytyy ehkä suorittaa tietyssä järjestyksessä. Esimerkiksi ottoautomaatti esimerkissä ei tililtä nostoa voi tehdä ennen kuin tilille on talletettu rahaa. Testitapaukset voi myös suunnitella taulukkomuotoon taulukon 2 mukaisesti.

Taulukko 2. Testitapaukset taulukkomuodossa

ID	KOHDE	ESIEHDOT	SYÖTE	ODOTETTU TULOS
Tt01	Tililtä nosto	saldo 200 €	20 €	nosto onnistui, uusi saldo:180 €
Tt02	Tililtä nosto	saldo 200 €	200 €	nosto onnistui, uusi saldo: 0 €
Tt03	Tililtä nosto	saldo 100 €	200 €	virheilmoitus: tilin saldo 100 €, ei voi nostaa

7.4 Testausmenettely

Testausmenettelyssä (Test Procedure Specification) kerrotaan miten testaus suoritetaan. Testausmenettelykuvaukseen on myös olemassa työkaluja. Dokumentti kannattaa pitää mahdollisimman yksinkertaisena. IEEE standardin 829-1998 mukaan testausmenettelymalli on seuraavanlainen: (Craig & Jaskiel 2002, 192–194.)

1) Testausmenettelyn yksilöivä tunnus (Test Procedure Specificaton Identifier)

2) Tarkoitus (Purpose)

Tässä kerrotaan testitapaukset, jotka suoritetaan ja kuvaillaan testin tarkoitus. Testitapauksiin voidaan viitata testitapauksen numerolla.

3) Erikoisvaatimukset (Special Requirements)

Tässä kerrotaan erityisehdot, joita testaus vaatii, jotta se voidaan suorittaa.

4) Testausvaiheet (Procedure Steps)

Tässä kerrotaan testausvaiheet vaihe vaiheelta. IEEE standardi pitää sisällään seuraavat stepit, jotka on esitelty taulukossa 3.

Taulukko 3. Testausmenettelyn testausvaiheet

STEPPI	NIMI	SELITYS
4.1	Loki	Testausten tulosten säilytys ja mitä tuloksille tehdään. Esimerkiksi testauksen tuloksia verrataan tuotantoajon tuloksiin.
4.2	Set up	Mitä toimenpiteitä tarvitaan, että testi voidaan aloittaa. Esimerkiksi ohjelma täytyy olla avattuna.
4.3	Start	Mitä toimenpiteitä tarvitaan, jotta testaus voidaan aloittaa. Esimerkiksi käyttäjä kirjautuu järjestelmään käyttäen salasanaa.
4.4	Proceed	Kuvaillaan kaikki työvaiheet stepeittäin, mitä testaus vaatii. Esimerkiksi 4.4.1 Valitse nosto 4.4.2 Anna summa 20 € 4.4.3 Valitse OK 4.4.4 Valitse lopeta jne…
4.5	Measure	Miten testitulosta arvioidaan. Esimerkiksi tilin saldoa vähennetään 20 € onnistuneen suorituksen jälkeen. Tämä voidaan todentaa tilin saldokyselyllä.
4.6	Shut down	Miten testaus keskeytetään, jos tarve vaatii. Esimerkiksi ohjelma lopetetaan log off-komennolla.
4.7	Restart	Määritellään kohdat, joista testauksen voi aloittaa uudelleen.
4.8	Stop	Mitä toimenpiteitä vaaditaan testauksen asialliseen lopettamiseen. Esimerkiksi yhteys palvelimeen katkaistaan.
4.9	Wrap up / restore	Millä toimenpiteillä ja mihin tilaan järjestelmä palautetaan.
4.10	Contingencies	Mitä toimenpiteitä tarvitaan, jos tapahtuu jotain odottamatonta. Esimerkiksi, jos tilillä ei ole rahaa, tilille täytyy suorittaa pano, joka on testitapaus Tt56.

8 Testauksen tulosten dokumentit

Testauksen suunnittelun jälkeen on aika tehdä itse testaus ja raportoida testauksen tuloksista. Testauksen suunnittelu ja valmistelu ajoittuu ohjelmistoprojektin koko elinkaareen, kun taas itse testaus ja testausten raportointi tyypillisesti ajoittuu ohjelmistoprojektin elinkaaren loppupäähän. Testatuista asioista pidetään kirjaa. Tulosteiden kirjaamiseen löytyy myös työkaluja. Alla on esitelty IEEE standardin mukaiset testausraporttimallit. Sekä suunnittelun puolella että myös tulosten dokumentoinnin puolella näitä malleja voidaan yhdistellä. Jos on kyse pienemmästä projektista, testauksen suunnitteludokumentteja voidaan käyttää tulosten dokumenttien pohjana. (Craig & Jaskiel 2002, 240–243.)

8.1 Testiloki

Testilokin (Test Log) tarkoituksena on tuottaa tietoa testaajille, käyttäjille ja kehittäjille sekä helpottaa testitapausten ongelmatilanteiden toistamista.
Testilokimalli:

1) Testin numero (Test Identifier number)
2) Kuvaus (Description)

Kuvaus siitä, mikä testi on kyseessä.

3) Toiminta ja tapahtuma merkinnät (Activity and Event Entries)

Esimerkiksi testattaessa online-toimintaa useammalla käyttäjällä, tähän merkataan testin aloitusaika, mahdollinen järjestelmän kaatuminen, järjestelmän toipuminen ja mitä muita toimenpiteitä on tehty.

Testilokin käyttäminen kannattaa pitää mahdollisimman yksinkertaisena, jotta sen päivittäminen olisi helppoa. Testilokin käytöstä tulee sitä tärkeämpi, mitä suurempi testattava järjestelmä on. (Craig & Jaskiel 2002, 244–245.)

8.2 Testauksen tapahtumaraportti / Virheraportti

Tapahtumaporttiin (Test Incident Report) kirjataan todelliset tulokset testien suorittamisesta ja kuvaukset virhetilanteista. Voidaan käyttää esimerkiksi taulukon 2. mukaista taulukkoa ja lisätä loppuun sarake: todellinen tulos, johon merkataan testauksen tulokset. Voidaan myös laatia erillinen testauksen tapahtumaraportti. (Craig & Jaskiel 2002, 248–251.)

Tapahtumaraporttimalli:

1) Testauksen tunnus (Incident Summary Report Identifier)

Testauksen tunnus.

2) Tapahtuman yhteenveto (Incident Summary)

Tässä viitataan testauksen dokumenttiin, jossa virhe huomattiin. Kaikista virheistä tulisi löytyä testattu testitapaus, testitapauksen tunnus. Jos virhe on huomattu ohi kirjattujen testitapausten, olisi löydetystä virheestä hyvä kirjoittaa testitapaus. Tämä helpottaa testaajia ja sovelluskehittäjiä myöhemmin, kun he testaavat tapausta uudelleen.

3) Tapahtuman / virheen kuvaus (Incident Description)

Virhe tulisi kuvata niin tarkasti, että raportin lukija ymmärtää, mistä on kysymys ja pystyy toistamaan virheen. Virheestä tulisi kuvata taulukon 4 mukaisia asioita.

Taulukko 4. Virheraportin kuvaus

STEPPI	NIMI	SELITYS
3.1	Syöte (Input)	Mikä on ollut todellinen syöteaineisto.
3.2	Odotettu tulos (Expected Results)	Mikä olisi pitänyt tulla tulokseksi.
3.3	Todellinen tulos (Actual Results)	Mitä todella tuli tulokseksi.
3.4	Poikkeavuudet (Anomalies)	Miten todellinen tulos eroaa odotetusta. Tähän voi myös kirjata, jos on jotain muuta poikkeavaa, kuten esimerkiksi kuukauden viimeinen päivä.
3.5	Päivä ja aika	Aika, milloin virhe tapahtui.
3.6	Testausmenettelyn steppi	Tarkalleen ottaen, missä stepissä testausmenettelyä virhe tapahtui. Tämä on tärkeää varsinkin silloin, jos on pitkiä testitapauksia käsittelyssä.
3.7	Ympäristö	Minkälaista testausympäristöä on käytetty.
3.8	Toistoyritykset	Kuinka monta kertaa testiä yritettiin toistaa.
3.9	Testaajat	Testin suorittajat.
3.10	Tarkkailijat	Kuka muu on tietoinen tilanteesta.

4) Vaikutus (Impact)

Kerrotaan mikä on virheen vaikutus. Riippuen siitä kuinka suuri virheen vaikutus on, se priorisoidaan suhteessa muihin virheisiin. Virheet vaikutuksen arvioimiseen voidaan käyttää virheen vakavuusasteen jakamista eri tasoihin esimerkiksi: pieni, suuri ja kriittinen. Tämän avulla on helpompi tehdä päätöksiä virheiden korjausjärjestyksestä. Suositus on maksimissaan viisi eri tasoa.

8.3 Testauksen yhteenvetoraportti

Yhteenvetoraportin (Test Summary Report) tarkoitus on koota yhteen testaustulokset ja tehdä niiden pohjalta loppuarvio. Raportissa kerrotaan myös sovelluksen mahdolliset rajoitukset tai virheiden todennäköisyys. Standardin mukaan, jos on tehty yleinen testaussuunnitelma ja eri tasoista omat testaussuunnitelmat, tulisi näistä kaikista tehdä myös testausyhteenveto.

Testauksen yhteenvetoraporttimalli:

1) Testauksen yhteenvetoraportin tunnus (Test Summary Report Identifier)

2) Yhteenveto (Summary)

Yhteenveto mitä testauksia tehtiin, ohjelmiston versio, testausympäristö yms. Tässä on myös viittaukset kaikkiin käytettyihin suunnitteludokumentteihin.

3) Poikkeamat suunnitellusta (Variances)

Kuvataan ne poikkeamat, joita oli testin suunnittelun ja itse testauksen välillä. Tämä auttaa testien suunnittelijaa parantamaan testisuunnitelmia jatkossa.

4) Testauksen kattavuus (Comprehensive Assessment)

Arvioidaan testauksen monipuolisuutta testaussuunnitelman kriteereihin nähden. Mainitaan piirteet, joita ei testattu riittävästi ja mahdolliset uudet riskit. Testauksen tehokkuuden mittarit samoin kuin niiden tulokset kerrotaan eli testauksen kattavuus.

5) Tulosten yhteenveto (Summary of Results)

Yhteenveto löydetyistä ja ratkaistuista virheistä sekä niistä virheistä, joihin ei löydetty ratkaisua.

6) Arviointi (Evaluation)

Testitapausten arviointi rajoitteineen. Arviointi perustuu testaustuloksiin ja niiden hyväksymis/hylkäys kriteereihin. Rajoitteiden kohdalla voidaan mainita esimerkiksi, että järjestelmää pystyy käyttämään 100 ihmistä kerrallaan.

7) Testaustoimintojen yhteenveto (Summary of Activities)

Yhteenveto testaustoimintojen päätapahtumista kuten resurssien käytöstä sekä testaukseen kuluneesta ajasta tärkeimmissä testaustapahtumissa. Yhteenveto on hyödyllistä tietoa suunniteltaessa tulevien testausten resursseja.

8) Hyväksyjät (Approvals)

Henkilöt, jotka hyväksyvät tämän raportin.

9 Ohjelmistotestauksen nykytilan arviointi

Perry (1995, 7–12) on tehnyt kysymyspatteriston, jonka avulla voidaan arvioida testausprosessin nykytilaa. Kysymyksiin vastataan joko kyllä tai ei. Epävarmassa tilanteessa vastataan ei. Jos jotain kohtaa ei sovelleta ollenkaan, vastataan: ei käytössä. Vastaamisen jälkeen lasketaan ei-vastaukset. Pisteet on jaettu karkeasti viiteen eri luokkaan ja niiden perusteella voidaan arvioida ohjelmistotestauksen nykytilaa, joka on esitetty taulukossa 3.

Kysymykset, joilla voidaan itse arvioida testauksen nykytilaa:

1. Onko joku yrityksessäsi vastuussa testauksesta?
2. Onko teillä ja käytättekö standardoitua testaussuunnitelmaa?
3. Onko teillä ja käytättekö standardoitua yksikkötestausta?
4. Onko teillä ja käytättekö standardoitua testausraportointia?
5. Meneekö teillä testisuunnitelma ja itse toteutusprosessi rinnakkain ohjelmistokehitysprosessin kanssa (testaus alkaa, kun kehitystyö alkaa ja loppuu, kun kehitystyö loppuu)?
6. Validoitteko, että määritykset ovat oikein toteutettu?
7. Validoitteko määritysten oikeintoteutuksen lisäksi, että asiakkaiden / käyttäjien odotukset on saavutettu?
8. Varmistavatko testaajat tilapäisten vaatimusten ja suunnittelun tarkkuuden ja täydellisyyden?
9. Raportoivatko testaajat virheistä työn tehneille sovelluskehittäjille (eivätkä toisille osapuolille, kuten johdolle)?
10. Tunnistavatko testaajat liiketoimintariskit ennen testaussuunnitelman kehittämistä?
11. Onko mitattavissa olevat testitavoitteet tunnistettu jokaiselle testattavalle ohjelmistolle?
12. Jos on, ovatko nuo mitattavat tavoitteet sidoksissa liiketoimintariskeihin?
13. Onko avoimista kirjatuista virheistä tehty yhteenveto ja käytetäänkö sitä testausprosessien ja kehityksen parantamiseen?
14. Onko testaajilla jo ennalta määriteltyjä virheiden odotuksia testattaessa perustuen aikaisempaan testauskokemukseen?
15. Onko teillä prosessia testausprosessin parantamiseksi?

16. Nimeättekö virheet?
17. Tekeekö yrityksesi yhteenvetoja ja käyttää virheellistä dataa testausprosessin tehokkuuden arvioimiseen tuottaakseen virheistä vapaata ohjelmistoa?
18. Teettekö tilastoja (kuten virheet / 1000 riviä koodia) suunnitellaksenne ja arvioidaksenne testiprosesseja?
19. Onko teillä prosessi testaajien kouluttamiseksi?
20. Onko automaattisten testityökalujen käyttö merkittävä osatekijä testiprosesseissanne?

Taulukko 3. Arvioinnin tulkinta (Perry 1995, 8–9)

Ei vastausten pistemäärä	**Arviointi**	**Arvioinnin selitys**
17-20	Testaus on kuin taidetta	Yrityksessä testaus on vahvasti riippuvainen yksittäisten testaajien tiedoista ja taidoista. Koulutusta ei järjestetä ollenkaan tai sitä on vähän. Testaustulokset voivat vaihdella erinomaisista huonoihin. Korkealaatuisten ohjelmistojen toimittaminen ei voi olla riippuvainen testauksen tehokkuudesta.
13-16	Testaus taitolajina	Yrityksessä on yleensä käytössä testausprosessi, joka pitää sisällään joitain standardeja ja testaussuunnitelman. Testaajat kuitenkin helposti poikkeavat testisuunnitelmasta ja keskittyvät virheiden etsintään
9-12	Testeissä käytetään määriteltyä testiprosessia	Yrityksessä on testiprosessit hyvin määriteltyjä, mutta eivät välttämättä tehokkaassa käytössä. Testausten lopussa ei yleensä tuoteta loppuraporttia testattavan ohjelmiston tilasta. Testaajien yleinen päätelmä on, että ohjelmisto joko on valmis laitettavaksi tuotantoon tai sitten ei.

5-8	Erinomainen testiorganisaatio	Yritys on keskittynyt selkeisiin testaustavoitteisiin ja optimoi testausresurssien käytön toteuttaakseen tavoitteet. Painopiste on koko testausprosessissa: analysoinnista testaustuloksiin ja puutteiden löytymiseen. Prosessia kehitetään noiden tulosten perusteella.
0-4	Maailmanluokan testiorganisaatio	Yrityksessä testaajat tekevät oikeastaan kaikki asiat oikein. Testaus perustuu riskien vähentämiseen, ne mitataan, testausprosessi on hyvin määritelty. Virheet kirjataan, analysoidaan, niistä tehdään yhteenveto ja niitä käytetään testausprosessin kehittämiseen. Testauskustannusten olisi pitänyt pienentyä huomattavasti. Asiakkaat luottavat testausprosessiin, joka tuottaa korkealaatuisia ohjelmistoja, sen sijaan että käytetäisiin hyväksymistestauksia apuvälineenä päätöksenteossa ohjelmiston toimivuudesta.

Ohjelmiston nykytilan arviointi kyseisen kyselylomakkeen avulla on hyvä lähtökohta, kun lähdetään miettimään testauksen ja siihen liittyvien dokumenttien kehittämistä yrityksessä. Kysymykset kattavat testaukseen liittyvät perusasiat. Tulosten jakautuminen viiteen eri luokkaan on riittävä taso karkean arvion suorittamiseksi. Suoritin tämän kyselyn esimiehelläni. Nykytilan arvioinnissa päädyimme luokkaan: testaus on kuin taidetta. Tulos omalta osaltaan puoltaa testauksen suunnitelmallisempaa käyttöönottoa.

10 Laskujen tulostuksen testaussuunnitelmien toteutus

Halusin lähteä liikkeelle ohjelmistotestauksen nykytilan arvioinnista. Testauksen nykytilan arvioinnin tulokset tukevat päätöstä aloittaa projekti testausprosessien kehittämisestä yhdessä ohjelmistotuotantoprosessien kanssa. Arvioinnin tulokset yhdessä opinnäytetyöni teoriaosuuden kanssa tukevat ohjelmistotestauksen kehittämisen aloittamista.

Koska aikaisempaa kokemusta testaussuunnitelman teosta ei ollut, lähdin testauksen teoriaan tutustumisesta liikkeelle. Päämääränä oli ymmärtää ja ottaa haltuun perusasiat testauksesta ja siihen liittyvistä dokumenteista. Testauksen suunnittelussa tarkoitus oli tutustua IEEE standardiin ja näiden mallipohjien kautta luoda testaussuunnitelma laskutuksen tulostukseen. Testaussuunnitelma on tehty puhtaasti teoriataustan pohjalta, koska käytännön kokemus järjestelmällisestä ja suunnitellusta testauksen suunnittelusta ja testauksesta puuttuvat. Kuten Craig ja Jaskiel (2002) teoksessaan Systematic Software Testing korostavat, testaussuunnitelman mallipohjat ovat hyvä yleisrunko, josta lähteä liikkeelle. Riippuen projektin laajuudesta, jokaiselle testaustasolle voi laatia oman testaussuunnitelman tai eri testaussuunnitelman mallipohjia voi yhdistellä. Tämä oli myös minun ajatukseni, kun lähdin suunnittelemaan testaussuunnitelmia.

Koska laskutuksen tulostusohjelman uudistus on luonteeltaan olemassa olevaan sovellukseen tehtävä yhden osa-alueen muutos, ei testaussuunnitelmia ollut järkevä tehdä jokaiselle tasolle erikseen. Päätin jakaa testaussuunnitelmat kahteen osaan: Yleiseen testaussuunnitelmaan ja moduulitestaussuunnitelmaan. Käytän jatkossa yleisestä testaussuunnitelmasta nimeä testaussuunnitelma. Laskujen tulostusohjelma on luonteeltaan hyvin yksityiskohtaista tiedon analysointia ja siksi tämä jako tuntui luonnolliselta.

Testaussuunnitelmassa on integrointitestauksessa testitapaukset laskuille haettavaa tulostustunnusta varten. Testauksen olisi voinut siirtää myös moduulitestaukseen. Päädyin tähän ratkaisuun, koska kysymyksessä on kahden eri aliohjelman yhdistäminen. Laskujen tulostustunnukset haetaan moduulitestauksen testitapausten tulosten perusteella. Testaussuunnitelmat ovat näin jaoteltuina selkeitä. Halusin myös luoda kattavamman mallipohjan integrointitestaukselle ja tässä tapauksessa tämä jako oli mielestäni perustelua.

Testitapausten valinnassa ei käytetty erikseen ekvivalenssiluokkiin jakamista, koska testit suoritettiin rinnakkaistestausperiaatteella. Testitapauksia tulee tällä periaatteella tarpeeksi jokaiseen ryhmään testausta varten. Testauksessa yksittäisiä testitapauksia verrataan tuotantoajon vastaaviin ja näin saadaan luotettava tulos testauksesta.

10.1 Testaussuunnitelma

Testaussuunnitelmaa laatiessani, lähdin liikkeelle testaussuunnitelman IEEE standardin 829-1998 mallipohjasta (Master Test Plan / Test Plan). Otin lisäksi mallia jo tehdystä testaussuunnitelmasta. Standardoidusta mallipohjasta on järkevää lähteä liikkeelle, koska näin kaikki asiat tulevat huomioitua paremmin. Sen pohjalta testaussuunnitelmaa on helppo lähteä toteuttamaan. Koska käytössäni ei ollut vaatimusmäärittely- eikä suunnittelun dokumentteja, käytin testitapausten suunnittelussa sekä ohjelmakoodia että laskutussovelluksen tuntemusta hyväkseni. Päämääränä oli tehdä yleinen suunnittelumalli noudattaen soveltuvin osin standardia. Testaussuunnitelmaan on yhdistetty integrointi- järjestelmä- ja hyväksymistestaus. Koska järjestelmä- ja hyväksymistestaustasojen rooli testauksessa on pieni, tuntui luontevalta yhdistää nämä kolme tasoa keskenään.

Testaussuunnitelmassa integrointitestauksen suunnittelun osuus oli merkittävin. Moduulitestauksen jälkeen integrointitestauksessa haetaan asiakkaan laskuille oikeat tulostustunnukset, joiden perusteella laskuille tulostuu vakiotekstit. Tämä sopi mielestäni integrointitestaukseen, koska laskutuksen tulostusohjelma koostuu kahdesta aliohjelmasta ja tässä moduulitestauksen tulostiedoston perusteella haetaan laskuille oikeat tulostustunnukset. Järjestelmä- ja hyväksymistestaustasot olivat käytännössä melkein samanlaisia. Halusin tässä kuitenkin ottaa ne erikseen, koska suunnittelen mallipohjaa, jota voi jatkossa hyödyntää ja koska halusin pitää hyväksymistestauksen virallisena tuotantoonotto-vaiheena.

Testaussuunnitelma pitää sisällään mainittujen tasojen testaussuunnitelman, testisuunnitelman, testitapausten määrittelyn ja testausmenettelyn. Ajojen käynnistys tapahtuu Windows Server ajoympäristössä oman käyttöliittymän avulla. Käyttöliittymän toiminnallisuutta ei tarvitse testata, koska muutoksia laskuajojen käynnistämiseen ei tullut. Testauksessa keskitytäänkin ohjelman toiminnallisuuden testaamiseen. Koska muutokset kohdistuivat laskutussovelluksen yhteen osa-alueeseen, laskujen tulostukseen, oli mielestäni järkevää yhdistää nämä kaikki testauksen suunnitteluun liittyvät dokumentit. Testaussuunnitelma on nyt yksinkertainen ja selkeä.

Lähdin liikkeelle testaussuunnitelmasta, täytin tarpeelliset kohdat ja lisäsin muiden testaussuunnitelmien mallipohjien asioita tarpeen mukaan. Seuraavaksi yksityiskohtaisemmin tekemäni ratkaisut.

Testaussuunnitelman mallipohjasta on laskutuksen testaussuunnitelmassa testaussuunnitelman tunnus ja versio. Versiohistoriasta mallipohjassa ei varsinaisesti puhuta mitään. Otin käytännön, jossa hyväksytty suunnitelma on versio 1.0. Tämä on mielestäni järkevä käytäntö jatkossakin. Version perusteella kaikki tietävät, milloin suunnitelma on hyväksytty. Sisällysluettelo on mukana, vaikka se ei IEEE standardiin kuulukaan. Mielestäni sisällysluettelo on tarpeellinen. Siitä näkee heti, mitä kaikkea suunnitelma pitää sisällään. Viittauksia ei tämän suunnitelman puitteissa ollut, joten jätin sen pois suunnitelmasta. Tilauksiin ja laskuihin kuuluu omaa termistöä niin paljon, että kokosin ne yhteen ja laitoin johdannon jälkeen kappaleeksi 2 Sanasto. Sanastokaan ei kuulu IEEE standardiin, mutta se on mielestäni tarpeellinen lisä mallipohjaan.

Johdannosta (kappale 1) yritin tehdä mahdollisimman yksinkertaisen ja selkeän. Testauksen kohteet (kappale 3) tein malliohjeen mukaan. Listasin suunnitelmaan ohjelmat, joita testataan. Viittauksia muihin laskutuksen dokumentteihin ei ollut käytössä, joten jätin ne pois. Ohjelmistoriskit (kappale 4) on mainittu yhdessä kappaleessa eikä eriteltynä, kuten Craigin ja Jaskielin (2002, 61) ehdottamassa mallipohjassa on. Tämän suunnitelman puitteissa yksi kappale ohjelmistoriskeihin riittää. Testattavat ominaisuudet (kappale 5) on kerrottu yleisellä tasolla, koska testitapaukset kuvataan tarkemmin testausmenettelyn yhteydessä, samoin ei-testattavat ominaisuudet (kappale 6). Testausstrategia testaustasoittain (kappale 7) on jaoteltu alatasoihin testaustasojen mukaan. Tässä pääpaino on integrointitestauksessa. Strategia osaan on koottu mallipohjan mukaisia asioita käytäntöön soveltaen. Jokaisesta testaustasosta on lyhyt kuvaus. Lisäksi kerrotaan testausmenetelmän käyttö, seuraavaan testaustasoon siirtymiskriteeri ja testaustyökalut.

Läpäisy- ja hylkäyskriteerit (kappale 8) on esitelty taulukkomuodossa. Keskeyttämisen ja jatkamisen perusteita en tähän suunnitelmaan kirjannut, koska en katsonut tätä osaa tarpeelliseksi tämän sovelluksen osalta. Testauksen raportointi (kappale 9) on tehty suoraan mallipohjan suosituksia noudattaen. Testaukseen liittyvät tehtävät olen yhdistänyt henkilöstö- ja koulutustarpeisiin. Mielestäni nämä voisivat jatkossakin olla samassa kappaleessa, eikä erikseen. Tes-

tausympäristö (kappale 10) ja vastuualueet (kappale 11) olivat selkeitä, eikä niissä ollut epäselvyyttä, samoin henkilöstö- ja koulutustarpeet (kappale 12). Aikataulun (kappale 13) tein laskutusprojektin aikataulun perusteella. Laskutusprojektin aikataulu muuttui sen verran, että vain osa testauksista on saatu tehtyä. Suunnitellut riskit ja niiden hallinta samoin kuin hyväksyjä ovat tästä suunnitelmasta pois. Projekti on yrityksen sisäinen eikä virallista hyväksyjää tarvita.

Testisuunnitelma, testitapaukset ja testausmenetelmä kuvataan kappaleesta 14 eteenpäin Testausmenettelyt otsikon alla. Testaustasot ovat kuvattuna erikseen. Näiden kaikkien dokumenttien tiedot on tässä yhdistetty. **Testisuunnitelmasta** on otettu mukaan testattavat ominaisuudet ja luotu testitapausten tunnukset. Tunnusten perusteella testitapauksia on helppo jäljittää. Tämä helpottaa myös testiraportteja kirjoitettaessa. Hyväksymis- ja hylkäyskriteerit on mainittu jo aikaisemmin, joten niitä ei enää tässä yhteydessä tarvitse mainita. Integrointitestauksessa testausvaiheet on koottu taulukkoon. Näin mielestäni testitapaukset ovat luettavassa muodossa. Kun kaikki testitapaukset tehdään saman mallin mukaan, niiden vertailu on helppoa.

Taulukossa on mainittuna testitapauksen tunnus, testauksen kohde, aloituksen edellytys eli syötteet ja odotettu tulos. Nämä ovat **testitapausten määrittelystä**. Laitteistovaatimukset on jo mainittu, joten jätin ne tästä pois. Erikoisvaatimukset ovat tässä suunnitelmassa tarpeettomia. Testitapausten väliset riippuvuudet on mainittu strategian yhteydessä (kappale 7.).

Testausmenettelyn mallista testitapauksissa on mainittu testauksen tarkoitus ja testausvaiheet. Testausvaiheet on kuvattu nimellä testitapaus. Lisäksi taulukossa on testitapauksen nimi ja testattavan testausvaiheen nimi. Testausvaiheen nimi on hyvä olla ainakin silloin, kun samassa testaussuunnitelmassa on testitapauksia eri testausvaiheille. Jos testaussuunnitelmia käyttää suoraan testiraporttien pohjana, on testausvaiheen nimi hyvä olla aina mukana. Testauksen raportointia varten mukana on valmiiksi testauspäivämäärä, testaaja, onko testi hyväksytty vai hylätty sekä huomiot testistä. Testausmenettelymallissa on mainittu muitakin asioita, mutta mielestäni tämän testaussuunnitelman puitteissa ne ovat tarpeettomia.

Laskutuksen tulostuksen testaussuunnitelma, joka on tehty A-lehtien laskutuksen tulostusohjelmiston uudistamisesta, on liitteessä 1.

10.2 Moduulitestaussuunnitelma

Laskutuksen tulostusprojektissa pääpaino oli yksittäisten moduulien testaamisessa ja siksi oli mielekästä laatia oma testaussuunnitelma moduulitestaustasosta. Ensimmäisessä versiossa tein yhden testaussuunnitelman, jossa oli kaikki testaustasot. Päädyin kuitenkin ratkaisuun, jossa erotin moduulitestauksen omaksi kokonaisuudeksi, koska testitapausten osuus on suhteellisen iso kokonaisuus tässä projektissa. Jaottelu on nyt selkeä ja testaussuunnitelmat ovat helposti luettavia. Moduulitestaus koostuu 21 alimoduulista, joissa haetaan asiakkaaseen, tilaukseen ja laskuun liittyviä tietoja. Näiden tietojen perusteella integrointitestauksessa haetaan laskulle oikea tulostustunnus. Moduulitestaussuunnitelmaa laadittaessa lähtökohdat olivat täsmälleen samat kuin testaussuunnitelmaa laadittaessa.

Moduulitestaussuunnitelmasta puuttuu sanasto, koska tässä osuudessa sanastoa ei tarvita. Tarvittavat tiedot on selitetty testausmenettelyn yhteydessä. Sisältö noudattaa muilta osin testaussuunnitelman sisältöä. Alilukuja on vähemmän, koska suunnittelussa on mukana vain yksi testaustaso. Testausmenettelyt-kappaleessa (kappale 13) ja testitapauksien selityksissä (kappale 13.2) testauksen vaiheet tulevat paremmin esille, koska yksittäisessä testitapauksessa haetaan useampi tieto yhdestä alimoduulista. Nämä testausvaiheet ovat laatikoissa testitapaus-nimellä. Testitapaukset ovat kuvattuina yksityiskohtaisella tasolla, koska testaajan täytyy tietää, mitä kyseinen alimoduuli tekee. Muilta osin moduulisuunnitelman suunnittelun perusteet ovat samoja kuin testaussuunnitelmankin. Nämä kriteerit ovat mainittuina opinnäytetyön luvussa 10.1. Testaussuunnitelma. Laskutuksen tulostuksen moduulitestaussuunnitelma on kuvattuna liitteessä 2.

10.3 Yleiset havainnot

Koska kyseessä oli olemassa olevaan sovellukseen tehtävän muutoksen testaaminen, ei testisuunnitelmien laatiminen mennyt kaikilta osin teoriassa esitellyn aineiston mukaan. Testauksen suunnittelussa tarvittavia dokumentteja ei ollut käytössä, vaan suunnittelu lähti liikkeelle koodiin tutustumalla. Suunnittelussa auttoi myös laskutussovelluksen tunteminen. Itse testaussuunnitelmien laatiminen lähti liikkeelle tyhjästä, koska testaukseen liittyviä dokumentteja ei ole yrityksessämme käytössä. Tämä antoi suunnittelulle haastetta, joka oli opinnäytetyön tarkoitus. Kun ensimmäinen konkreettinen testauksen mallipohja on olemassa, sitä on helppo lähteä työstämään yhdessä eteenpäin.

11 Pohdinta

Tämän opinnäytetyön tavoitteena oli tutustua testauksen osuuteen ohjelmistoprojekteissa. Mitä, miten ja milloin ohjelmistotestausta tehdään. Testaukseen tutustumisen kautta päätavoitteena oli luoda testauksen suunnittelumallit, joita voimme A-lehdissä tulevissa ohjelmistoprojekteissa hyödyntää. Henkilökohtaisena tavoitteena oli tämän opinnäytetyön puitteissa saada hyvä yleiskäsitys testauksesta ohjelmistoprojekteissa.

11.1 Tavoitteena testaussuunnitelma

A-lehdissä ei ole käytössä yhtenäistä, suunnitelmallista testauksen projektimaista etenemismallia siihen liittyvine dokumentteineen. Suuremmissa projekteissa, joissa on ollut kyse uudesta ohjelmistosta, on ollut käytössä etukäteen suunniteltuja testauksia. Pääasiassa sovelluskehitystä olemassa oleviin ohjelmistoihin on tehty uusien piirteiden lisäyksinä tai olemassa olevien piirteiden muutoksina. Näissä testausta suoritetaan suunnittelijoiden toimesta niin, että jokainen testaa omat ohjelmansa. A-lehdissä on jokin aika sitten otettu käyttöön projektimalli ja testauksen liittäminen tähän projektimalliin omana kokonaisuutenaan olisi luonteva jatko ohjelmistoprojektin ja testauksen prosessien kehittämiselle.

Vaikka testaussuunnitelmat on suunniteltu tuotannossa olevan sovelluksen osaan tehtävästä uudistamisesta, onnistuin mielestäni luomaan hyvän pohjan suunnittelumallille. Suunnittelun haasteena oli tehdä kattava testaussuunnitelma niin, että kaikki tarpeelliset osa-alueet on huomioitu. Haasteena oli myös yhdistää eri testaussuunnitelmien mallipohjista olennainen testauksessa tarvittava tieto. Koska testattava alue oli suhteellisen pieni, suunnitelman tiivistäminen oli järkevää. Mielestäni suunnitelman täytyy olla tarpeeksi yksinkertainen, jotta se pysyy luettavana ja mielenkiintoisena. Liian pitkät testaussuunnitelmat eivät innosta projektin jäseniä tutustumaan suunnitelmiin. Mielestäni tämä suunnittelumalli oli myös hyvä esimerkki eri testausmallien yhdistämisestä. Vaikka standardit pitävät sisällään monta eri dokumenttia, ei niitä kaikkia ole järkevä käyttää pienissä projekteissa. Viimeaikaiset projektit A-lehdissä ovat olleet pienimuotoisempia ja silloin on järkevää käyttää yksinkertaista ja selkeää suunnittelun kuvausta. Testauksesta ei pidä tehdä liian raskasta, mutta täytyy myös antaa arvoa testauksen hyödyille. Kun projektit ovat suuria ja riskienhallinta kasvaa, on järkevää ottaa käyttöön eri testaustasoille omat suunnittelumallit.

Käytännössä testaussuunnitelmat täytyy miettiä jokaiseen projektiin erikseen, mutta tekemäni mallipohjat ovat hyvä lähtökohta. Pienemmissä projekteissa voi mallipohjana käyttää tekemääni testaussuunnitelmaa, jossa eri tasoja on yhdistelty. Suuremmissa projekteissa, joissa testaussuunnitelmia tehdään eri testaustasoille erikseen, voi mallipohjana käyttää tekemääni moduulitestaussuunnitelmaa. Näiden mallipohjien tukena on testauksesta tekemäni teoriaosa, jossa eri suunnittelun mallipohjat on selitetty yksityiskohtaisesti. Teoriaosa tukee myös muilta osin testaussuunnitelmien tekoa ja testitapausten valintaa. Itse laskutuksen projektin aikataulumuutoksista johtuen testauksen toteutus viivästyi sen verran, että en ehtinyt saada tähän opinnäytetyöhön testauksesta enkä siihen liittyvistä dokumenteista aineistoa. Ohjelman suunnittelijan tehdessä testausta, ovat testaussuunnitelmat lähinnä testauksen tukena. Muun henkilön kuin ohjelman suunnittelijan testatessa ohjelmaa, on tekemistäni testaussuunnitelmista mielestäni hänelle hyötyä, koska se on niin yksityiskohtainen ja kattaa laskutuksen tulostuksen kaikki vaiheet.

Jatkokehitystä suunniteltuihin malleihin tarvitaan. Ensimmäiseksi tarvitaan yrityksen johdolta päätös testauksen kehittämisestä. Päätöksen jälkeen suunnitellaan testausprosessi ja testauksessa käytettävät dokumenttien mallipohjat. Tekemäni mallipohjat ovat hyvä lähtökohta ja nopeuttavat mallipohjien suunnittelua. Myös opinnäytetyön teoriaosuudesta on hyötyä testausprosessia, testitapauksia, testausmenetelmiä ja testauksen mallipohjia suunniteltaessa. Suunnitelmat kannattaa mielestäni tehdä pienryhmässä, johon kuuluu noin kolme henkilöä. Käytännön työ ja testausraportit antavat hyödyllistä tietoa ja auttavat tulevien testausten suunnittelussa ja testauksessa. Testauksessa käytettäviin mittareihin tulee kiinnittää huomiota. Oikein valitut mittarit nopeuttavat testaamista, eikä turhia testauksia tehdä. Raportoinnin avulla testauksesta ja ohjelmistoprojektista tulee läpinäkyvää ja näin projektin jäsenet ovat ajan tasalla tilanteesta. Raporttien avulla voidaan myös jakaa tietoa projektin tilanteesta projektin ulkopuolisille jäsenille, esimerkiksi yrityksen johdolle. Testaus kannattaa ehdottomasti ottaa mukaan ohjelmiston tuotantoprosessiin omana kokonaisuutena ja toivon, että tämä opinnäytetyö osaltaan vakuuttaa lukijat testauksen tärkeydestä ohjelmistoprojekteissa.

11.2 Muut tavoitteet

Testauksen suunnittelun aloittaminen ohjelmistoprojektin alussa heti määrittelyvaiheessa, on tärkeää ajatellen laadukasta ohjelmistoa, joka saavutetaan mahdollisimman kustannustehokkaasti. Staattisia testausmenetelmiä ei aina ole mielletty itse testaukseen kuuluvaksi, mutta nii-

den käyttö katselmusten ja tarkastusten muodossa luo ohjelmistoprojektille jo alusta lähtien tehokkaan lähtökohdan. Jo projektin varhaisessa vaiheessa huomatut virheet ja niiden korjaukset säästävät kustannuksia huomattavasti. Tarkastusten kautta saadaan myös uuden ohjelmiston tietotaitoa levitettyä varhaisessa vaiheessa projektin jäsenille ja tuleville käyttäjille. Tätä kautta siirryttäessä testauksen suunnittelussa seuraaville tasoille, suunnittelutyö helpottuu. Jotta testauksen suunnittelutyö etenisi aikataulun mukaan, on tarvittavien dokumenttien itse projektiin liittyen oltava ajoissa saatavilla. Näitä ovat projektisuunnitelma, vaatimusmääritykset, käyttöohjeet, arkkitehtuuri- ja moduulisuunnitelmat.

Yleisin käytetty testausmalli on V-malli. Siinä testaus jaetaan eri testaustasoihin (moduuli-, integrointi-, järjestelmä- ja hyväksymistestaus). Testauksen suunnittelu ja testaus etenevät tämän vaihejakomallin mukaan rinnakkain ohjelmistoprojektin kanssa. Testaus aloitettuna heti ohjelmistoprojektin määrittelyvaiheessa parantaa riskien hallintaa ja tuottaa kustannustehokkaampia ohjelmistoja.

Testausmenetelmät jaetaan staattisiin ja dynaamisiin testausmenetelmiin. Dynaamisista testausmenetelmistä yleisimmät ovat mustalaatikko- ja lasilaatikkotestaus. Testitapausten valinnassa kannattaa kiinnittää huomiota mahdollisimman kattavaan testitapausten valintaan. Mustalaatikkotestauksessa testaamista helpottaa ekvivalenssiluokkien käyttö, jossa testitapaukset jaetaan luokkiin niiden ominaisuuksien mukaan. Lasilaatikkotestauksessa tavoitteena on ohjelman kaikkien haarojen ja ohjelmapolkujen läpikäynti. Testattavan aineiston tulisi olla mahdollisimman kattava. Kattavuutta mitataan eri tavoilla ja nämä tavat määräävät testiaineiston valintaa. Koodikattavuutta pidetään tärkeimpänä kattavuuden mittana. Koodikattavuutta mitataan esimerkiksi lause- ja polkukattavuudella. Muita testausmenetelmiä ovat mm. harmaalaatikko-, top-down- ja bottom-up-testaus. Ohjelmistoa testattaessa täytyy kiinnittää huomiota paitsi virheiden etsimiseen myös ohjelman käytettävyyteen, suorituskykyyn ja luotettavuuteen.

Testaussuunnitelman tarkoitus on löytää kaikki ne tekijät, joita ohjelmistoprojektissa halutaan testata. Testaussuunnitelma vastaa kysymyksiin mitä, miten ja milloin sekä millaisia lopputuloksia odotetaan. Testaussuunnitelman tarkoitus on järjestää testaus eri vaiheisiin, jolloin testaus etenee järjestelmällisesti eteenpäin yhdessä ohjelmistoprojektin kanssa. Ohjelmistotestauksen onnistumiseen vaikuttavat sekä motivoituneet ja ammattitaitoiset testaajat että hyvin

laaditut testauksen dokumentit, joita ovat testaussuunnitelmat ja testauksen tulokset. Dokumentit auttavat myöhemmin laadittaessa uusia testaussuunnitelmia.

Testauksessa käytettävät raportit IEEE 829-1998 standardin mukaan ovat: testaussuunnitelma, testisuunnitelma, testitapausten määrittely, testausmenettely, testiloki, testauksen tapahtumaraportti ja testauksen yhteenvetoraportti. Testaussuunnitelman voi laatia kaikille tasoille erikseen tai, kun kyse on pienemmästä ohjelmistoprojektista, eri testaussuunnitelmia voi yhdistellä.

Opinnäytetyö antoi vastaukset kysymyksiin, joita tavoitteenani oli selvittää. Haasteellisinta oli siirtää teoria käytäntöön testaussuunnitelmien teossa. Opinnäytetyö on tiivis kokonaisuus, jossa on keskitytty testauksen perusasioihin ja testaussuunnitelmien tekoon. Seuraava vaihe on siirtää teoria käytäntöön, tutustua testaukseen ja testauksen raportointiin ja tätä kautta kehittää testausprosessia.

12 Yhteenveto

Opinnäytetyössä käytiin läpi perusasioita testauksesta ja testauksen suunnittelusta. Testaussuunnitelmien teko parhaillaan menossa olevaan projektiin ovat näiden perusasioiden tuotos. Opinnäytetyö antaa mielestäni näihin hyvät vastaukset ja opinnäytetyön tavoitteet täyttyivät. Testaus ja testauksen liittyvät dokumentit ovat avainasemassa onnistuneessa ohjelmistoprojektissa. Testauksen avulla saadaan laadukas ohjelmisto käyttäjille. Testauksen raportointi auttaa uusien ohjelmistoprojektin toteuttamisessa, kun raportoinnista saadaan tietoa haasteellisista ja yleisimmistä ongelmakohdista. Testaus nopeuttaa myös ohjelmistoprojektin onnistumista, kun virheet löydetään aikaisessa vaiheessa. Ilman dokumentointia tämä ei ole mahdollista.

Opinnäytetyö antaa perustietoa testauksen suunnitteluun. Opinnäytetyössä olevien mallipohjien ja kuvausten avulla voidaan testauksen suunnittelu aloittaa. Opinnäytetyössä oleva teoriaosa tukee testaussuunnitelmien tekoa ja testitapausten valintaa.

Testausprosessien kehittäminen tulisi mielestäni aloittaa A-lehdissä. Testauksen suunnitelmallinen eteneminen on erittäin tärkeää onnistuneessa ohjelmistoprojektissa. Testauksesta olisi hyvä suunnitella oma kokonaisuus A-lehdissä jo käytössä olevan ohjelmistoprojektin rinnalle niin, että siitä tuotetaan omat dokumentit. Testauksen kehittämiseen olisi hyvä perustaa pieni ryhmä, joka suunnittelee testausprosessin ja siinä käytettävät dokumentit.

Lähteet

Craig, R. & Jaskiel, S. 2002. Systematic Software Testing. Artech House Publishers. London.

Enqvist, A. & Krats, A. & Lahti, K. & Luukkonen, M. & Maanselkä, A. & Yli-Honkola, J. 2001. Ohjelmiston testaus. Ohjelmisto-tuotantokurssin harjoitustyö. Jyväskylän yliopisto. Luettavissa: http://webcache.googleusercontent.com/search?q=cache:IhDZnmfxz-AJ:ftp://85.118.226.202/Documentation/%25D2%25E5%25F5%25ED%25E8%25F7%25E5%25F1%25EA%25EE%25E5/%25CA%25EE%25EC%25EF%25FC%25FE%25F2%25E5%25F0%25ED%25EE%25E5/Books/Software%2520Engineering/R_10.pdf+Enqvist+ohjelmiston+testaus&hl=fi&gl=fi. Luettu 25.9.2010.

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. 11. painos. Talentum. Helsinki.

IEEE standardi 829-1998. Luettavissa: http://standards.ieee.org/reading/ieee/std_public/description/se/829-1998_desc.html. Luettu 18.9.2010.

Jäntti, M. 2003. Testitapausten suunnittelu UML-mallinnuksen avulla. Pro Gradu tutkielma. Tietojenkäsittelytieteen laitos. Kuopion yliopisto. Luettavissa: www.cs.uku.fi/research/Teho/mjanttigradu.pdf. Luettu 19.9.2010.

Partanen, P. 2009. Laadukkaan ohjelmistotestauksen piirteet. Tutkintotyö. Tampereen ammattikorkeakoulu. Luettavissa: http://publications.theseus.fi/handle/10024/10184. Luettu: 26.9.2010

Perry, W. 1995. Effective Methods for Software Testing. Katherine Schowalter. Canada

Rajamäki, J. 2000. Ohjelmiston testaus. Kurssimoniste. Helsingin ammattikorkeakoulu. Luettavissa: http://www.google.fi/search?q=jussi+rajam%C3%A4ki+ohjelmiston+testaus&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:fi:official&client=firefox-a. luettu 26.9.2010.